Proyecto de Sistemas Informáticos

Curso 2005-2006

Aplicación de Técnicas de CBR a los Sistemas de Enseñanza Interactivos

Ignacio Iglesias Franch
Denis Jiménez Requero
Javier Luengo Requero

Dirigido por:
Belén Díaz Agudo
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

AUTORIZACIÓN A LA UNIVERSIDAD

Nosotros Denis Jiménez Requero, Javier Luengo Requero e Ignacio Iglesias Franch autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmas:

Ignacio Iglesias Franch Denis Jiménez Requero Javier Luengo Requero

En Madrid a 04 de Julio de 2006

Proyecto de Sistemas Informáticos

Curso 2005-2006

Aplicación de Técnicas de CBR a los Sistemas de Enseñanza Interactivos

Ignacio Iglesias Franch
Denis Jiménez Requero
Javier Luengo Requero

Dirigido por:
Belén Díaz Agudo
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

RESUMEN DEL PROYECTO

El objetivo de este proyecto ha sido la especificación, diseño e implementación de un sistema de enseñanza interactiva usando técnicas de razonamiento basado en casos, en inglés "Case Based Reasoning" **[CBR].** Un sistema de enseñanza interactivo es una aplicación capaz de proponer ejercicios a un usuario y ayudar a este a resolverlos. No se limita a proponer un ejercicio y mostrar la solución como cualquier libro, sino que aporta las ayudas apropiadas durante la resolución del ejercicio propuesto, además es capaz de procesar una pregunta concreta sobre un término del dominio.

PROJECT SUMMARY

The objective of this project has been the specification, design and implementation of an interactive learning system using Case Based Reasoning **[CBR]** techniques. An interactive learning system is an application capable of proposing exercises to the user, helping him solve them. It's not limited to proposing exercises and showing the answer like any book would do, it displays appropriate help during the solving process, furthermore it's capable of processing a concrete question about a term of the knowledge domain.

ACRÓNIMOS Y PALABRAS CLAVE

Software (SW)
Inteligencia Artificial
Case Base Reasoning (CBR) o Razonamiento basado en casos.
Java Virtual Machina (JVM)
Learning by doing

PRÓLOGO

La motivación principal de este proyecto es la creación de un producto Software (SW) para la enseñanza. La informática es una ciencia muy amplia, que se puede aplicar cualquier campo del conocimiento humano, desde la medicina hasta el derecho o la psicología. El campo de la docencia es uno de los menos explotados, en concreto la creación de un sistema capaz de enseñar cualquier materia. **[JVM], [C++].**

Otra de las motivaciones de este proyecto es el uso de la Inteligencia Artificial (IA), la cual está en auge y cada vez posee más ramas abiertas para la investigación. La IA intenta analizar y comprender los mecanismos que dan lugar a conductas inteligentes para, a partir de ahí, reproducir dichas conductas en máquinas (no necesariamente con los mismos mecanismos). **[Nelson & Richard Hollingham]**

El desarrollo de un producto SW, no es sólo la implementación del código para su correcto funcionamiento, sino que hay mucho más. Durante un tiempo las empresas contrataban a gente que supiera programar para desarrollar un producto, pensando que eso les ahorraba dinero, pero la realidad bien distinta. Puesto que la informática avanza muy rápidamente, los productos deben adaptarse al cambio para no quedarse obsoletos, si el desarrollo del sistema no posee una buena documentación o el código no está pensado para dicho cambio, actualizarlo será casi más caro que crear un producto nuevo. La ingeniería del SW define las tareas necesarias para evitar estos problemas, y por ello el desarrollo del sistema que se detalla a continuación se basa en distintas partes de la ingeniería del SW.

INDICE

CAPÍTULO 1
INTRODUCCIÓN

1. INTRODUCCIÓN

1.1 DESCRIPCIÓN

Actualmente cuando alguien se plantea aprender una asignatura práctica suele utilizar un libro de ejercicios en el que puede o no venir la solución de los mismos. Este método de aprendizaje es repetitivo y poco intuitivo ya que el interesado tiene que comparar las soluciones con sus ejercicios y buscar los errores, además consultar la documentación para saber por que ha cometido el error. Por ello nos planteamos la creación de un sistema de enseñanza interactivo.

Un sistema de enseñanza interactivo es una aplicación capaz de proponer ejercicios a un usuario y ayudar a este a resolverlos. No se limita a proponer un ejercicio y mostrar la solución como cualquier libro, sino que aporta las ayudas apropiadas durante la resolución del ejercicio propuesto, además es capaz de procesar una pregunta concreta sobre un término del dominio, obviamente características carentes en los libros. El sistema se ha basado, por tanto, en el paradigma de aprendizaje con ejercicios "Learning by doing" **[LBD]**, ya que creemos es un método muy efectivo, y más aún si se intercala con ayudas procedentes de la documentacón .

El objetivo inicial del proyecto es un sistema independiente del dominio. Este proyecto ha sido propuesto por GAIA (Group for Artificial Intelligence Applications) **[GAIA]**, perteneciente al departamento de Sistemas Informáticos y Programación **[SIP]**, como una extensión del entorno JAVY (JavA taught Virtually) **[JAVY],** un aplicación basada en la metodología "Learning by doing" **[LBD]** donde los estudiantes pueden aprender el lenguaje de la maquina virtual de JAVA **(Capítulo 4.3) [JVM]** y su estructura utilizando un entorno metaforico 3-D que simula la JVM. **[JVM].** Por ello se ha utilizado como dominio para este sistema de enseñanza interactivo el lenguaje de la maquina virtual de JAVA. En el capitulo 7.2 se describe como aplicar la arquitectura de este proyecto a otros dominios.

Por tanto el sistema consta como mínimo de dos usuarios, uno que trata de aprender un dominio de conocimiento (alumno) y otro con conocimiento del domino que trata de enseñar (profesor).

El funcionamiento del sistema tiene dos partes fundamentales:

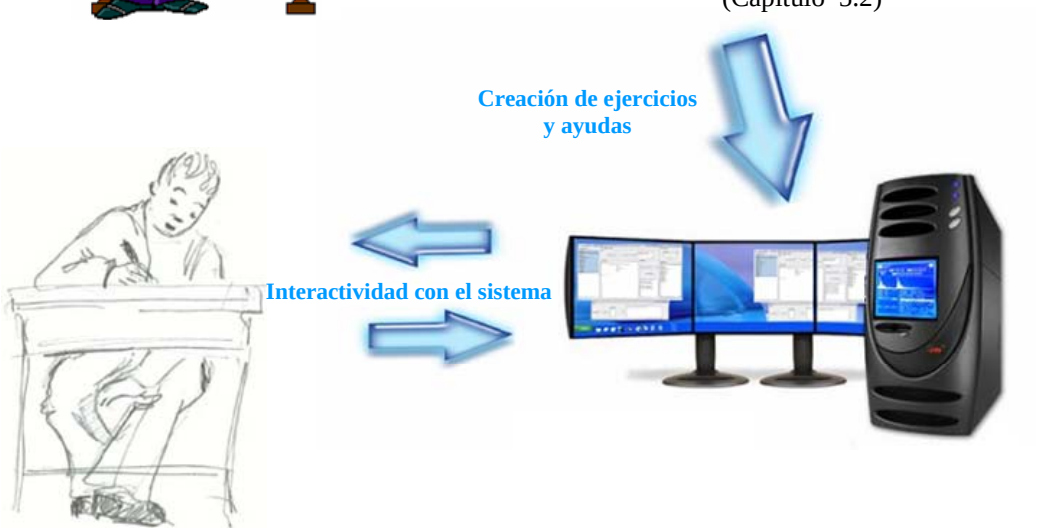
- El profesor especifica el dominio añadiendo la ontologia del conocimiento **(Capítulo 3.2**) **[ONT],** la documentación textual del dominio **(Capítulo 3.3.3) [JVM],** y es el encargado de crear los ejercicios y las ayudas gracias al editor de ejercicios **(Capítulo 5.2)**, estos ejercicios serán propuestos para su resolución al alumno por el módulo de aprendizaje **(Capítulo 5.3**).

- El alumno simplemente tendrá que intentar resolver los ejercicios que el módulo de aprendizaje le propone y en el caso que cometiera errores en la resolución el sistema le ofrecerá ayudas para que los resuelva. O incluso el alumno podrá preguntar directamente al sistema de forma textual sobre la información de la JVM **[JVM]**

En la figura 1.1 se muestra el funcionamiento de el sistema que se acaba de describir.

Diagrama del sistema de enseñanza interactivo

Profesor determina un dominio concreto del conocimiento:
- ✓ Documentación Textual (Capítulo 5.5)
- ✓ Ontología (Capítulo 3.2)
- ✓ Creación de ejercicios y ayudas(**Editor de ejercicios**) (Capítulo 5.2)



Alumno:
Recibe ayudas de distintos tipos mientras intenta resolver el ejercicio **(Módulo de aprendizaje)** (Capitulo 5.3):
- ✓ Textuales (Capitulo 5.5)
- ✓ Sobre errores cometidos (Capitulo 5.4)
- ✓ Documentación del dominio (Capitulo 4.2.2)

Figura 1.1: Esquema del sistema de enseñanza interactivo que muestra el funcionamiento del mismo

1.2 OBJETIVOS INICIALES

Dos de los propósitos que se buscan en este apartado son, por un lado tener una visión general del producto que queremos desarrollar, y por otro poder analizar los objetivos cumplidos al terminar la aplicación.

El objetivo inicial es la creación de un sistema de enseñanza interactivo. Para ello se plantean las siguientes expectativas:

- Gestión sobre un conocimiento especifico. La aplicación debe enseñar a los alumnos sobre el conocimiento del dominio.

- Interacción con distintos tipos de usuario, alumnos y profesores.

- Gestión de los alumnos mediante perfiles de usuario que definan el nivel de aprendizaje del usuario.

- Generación automática de ejercicios para la enseñanza. El método de aprendizaje se basa en la propuesta de determinados ejercicios según el perfil de usuario.

- Capacidad de aprendizaje por parte del sistema, para proponer ayudas al usuario no previstas inicialmente.

- Interfaz gráfico intuitivo. Se busca que los usuarios no necesiten un periodo de aprendizaje para el manejo de la herramienta final.

- Sistema capaz de ayudar al alumno de forma reactiva (el alumno solicita ayuda y el sistema la genera) y proactiva (el sistema es capaz de determinar cuando el alumno necesita ayuda para continuar resolviendo el ejercicio).

CAPÍTULO 2
FASE DE ANÁLISIS

2. FASE DE ANÁLISIS

2.1 INTRODUCCIÓN

El objetivo de este apartado es obtener un diseño que identifique los distintos módulos que intervienen en el sistema, especificando la funcionalidad de los mismos con poco nivel de detalle.

A partir de los objetivos iniciales planteados en el apartado anterior definimos los requisitos o características de la aplicación. Una vez especificadas las características, podremos definir los módulos que van a intervenir en el sistema.

2.2 ESPECIFICACIÓN DE REQUISITOS

2.2.1 Introducción

El objetivo es describir las características del sistema, determinar las funcionalidades y las restricciones del mismo.

El tener un documento de especificación de requisitos aporta muchas ventajas, tanto durante el desarrollo de la aplicación como durante su mantenimiento **(Capítulo 7)**. Ayuda a elaborar el diseño del sistema **(Capítulo 5)** y a implementarlo **(Capítulo 6**).

2.2.2 Requisitos funcionales

Son los que determinan las funcionalidades del sistema.

- Posee un editor de ejercicios para la creación de los mismos.
- Permite almacenar los ejercicios.
- Permite abrir ejercicios desde un fichero para editarlos si fuese necesario.
- Creación de perfiles de usuario que contiene el nivel de conocimiento del alumno y la lista de ejercicios que ya ha solucionado correctamente.
- Proposición selectiva de ejercicios, es decir, un alumno puede elegir el ejercicio que quiere solucionar de entre todos los ejercicios.
- Permite almacenar los perfiles de usuario, para que estos no se eliminen al cerrar la aplicación.
- Permite eliminar los perfiles de usuario cuando el profesor lo crea oportuno.
- Proposición automática de ejercicios, es decir, el sistema propone un ejercicio al alumno en función a su perfil de usuario.
- Sistema capaz de ayudar al alumno. Cuando un alumno solicita ayuda el sistema debe ser capaz de determinar cuál es el problema y ayudarle a resolverlo.
- El sistema aprende para que las ayudas generadas sean cada vez más adecuadas.
- El sistema es capaz de acceder al conocimiento que trata de enseñar.

- Proporciona distintos tipos de ayuda, textual, gráfica (paginas HTML de referencia) **[JVM],** o bien ayuda con la resolución del ejercicio.

2.2.3 Requisitos no funcionales

Hace referencia a características tecnológicas y de diseño, sin que aporten ninguna información sobre el funcionamiento de la aplicación.

- El conocimiento está separado del razonamiento.

- El sistema es multiplataforma.

- La parte gráfica debe estar claramente separada de la implementación asociada a cada elemento gráfico. Este requisito es importante para poder cambiar el interfaz gráfico con facilidad.

- La gestión de ayudas está implementada mediante un sistema de razonamiento basado en casos (**Capitulo 3.3**).

2.2.4 Interfaces de usuario

En este apartado se describen los requisitos que cumplen los interfaces de usuario.

- Son claros, y sencillos. El realizar una acción por parte del usuario no le tiene que llevar mucho tiempo, debe ser capaz de entender el funcionamiento del interfaz mediante un vistazo al mismo. Los usuarios no necesitarán un curso formativo para el manejo de la herramienta.

- Poseen ayuda. Para aquellas tareas más complicadas de manejar, el interfaz propone una breve descripción que explica al usuario qué debe hacer.

- Las acciones más comunes de un usuario están más accesibles que aquellas que se usen en menor frecuencia.

- El interfaz da soporte gráfico a los requisitos funcionales de cada usuario. Todas aquellas acciones que pueda realizar el profesor o el alumno deben realizarse a través de elementos gráficos.

2.3 CASOS DE USO

Una parte fundamental a la hora de realizar el análisis es ver la funcionalidad del sistema desde el punto de vista del usuario, para ello se especifican los casos de uso para cada uno de los usuarios. Es decir, se describe un uso del sistema y como este interactúa con el usuario.

Para describir un caso de uso es necesario identificar a los actores que intervienen en el sistema. Un actor es un usuario de la aplicación, o un sistema externo que interactúe con la aplicación.

Se identifican los siguientes actores:

- Profesor
- Alumno

2.3.1 Casos de uso del alumno.

En primer lugar el alumno tiene que cargar su usuario o perfil. En caso de que no existiera, se crea un perfil nuevo.

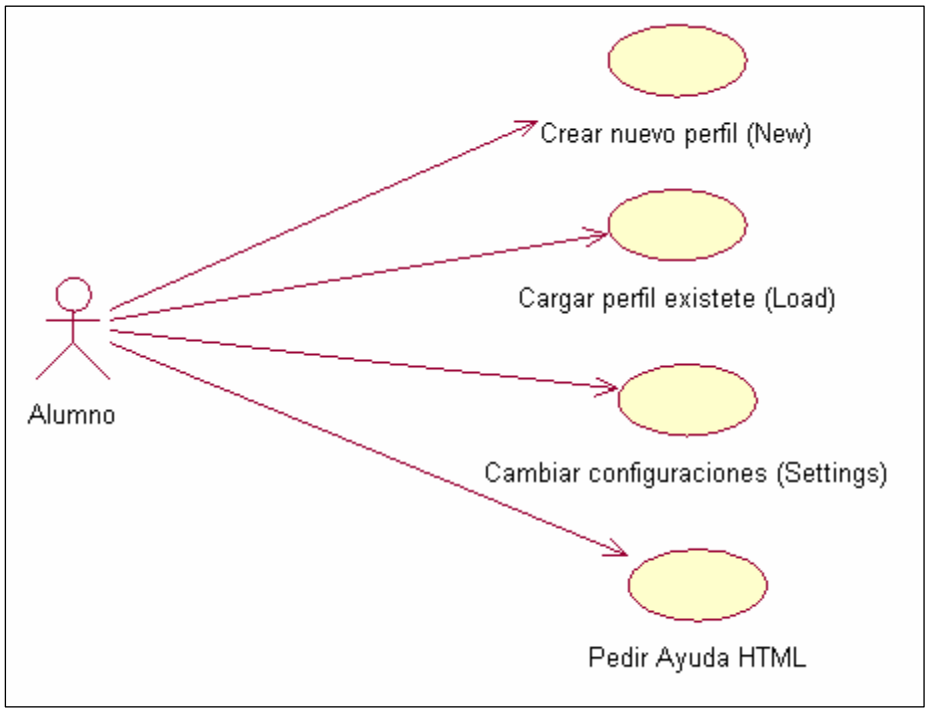


Figura 2.1: Casos de uso del alumno.

- Caso de uso "Crea nuevo perfil"

  - Objetivo: Creación de un perfil para poder utilizar la herramienta de enseñanza. Y así poder cargar en nuevas ocasiones el perfil.

  - Entradas: Datos del usuario: DNI o identificador de usuario.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje. No hay ninguna restricción para el usuario, todos pueden acceder. En el caso de que el usuario no esté registrado se le informará y se abrirá el formulario de registro.

- Salida: Se habilitan todas las funcionalidades y botones del interfaz de usuario.

- Post-condiciones:

    - Condiciones de final exitoso: El actor puede utilizar todas las prestaciones de la herramienta.

    - Condiciones de final fallido: El actor no puede utilizar todas las prestaciones de la herramienta y deberá crear un nuevo perfil o cargar uno existente para poder acceder correctamente.

- Actor primario: Alumno.

o Caso de uso “Carga perfil existente”

- Objetivo: Carga de un perfil de usuario en el que están almacenados los avances conseguidos (ejercicios resueltos y demás).

- Entradas: Datos del usuario: DNI o identificador de usuario.

- Precondiciones: Se debe haber iniciado el módulo de aprendizaje. No hay ninguna restricción para el usuario, todos pueden acceder. En el caso de que el usuario no esté registrado se le informará y se abrirá el formulario de registro.

- Salida: Se habilitan todas las funcionalidades y botones del interfaz de usuario.

- Post-condiciones:

    - Condiciones de final exitoso: El actor puede utilizar todas las prestaciones de la herramienta.

- Condiciones de final fallido: El actor no puede utilizar todas las prestaciones de la herramienta y deberá crear un nuevo perfil o cargar uno existente para poder acceder correctamente.

- Actor primario: Alumno.

- Caso de uso “Pedir Ayuda HTML”

  - Objetivo: Mostrar al usuario la ayuda de la aplicación y del libro de la máquina virtual de Java.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje.

  - Salida: Se muestra el libro de la JVM en el explorador de Internet por defecto del computador del usuario.

  - Post-condiciones:

    - Condiciones de final exitoso: Se abre el explorador de Internet y se muestra el libro en HTML de la JVM.

    - Condiciones de final fallido: Si no esta bien configurado la aplicación encargada de abrir los archivos HTML en el computador del usuario, podría abrirse con otra aplicación en formato texto , o pedir que se seleccione una de una lista de aplicaciones.

  - Actor primario: Alumno.

- **Casos de uso una vez cargado un perfil**:

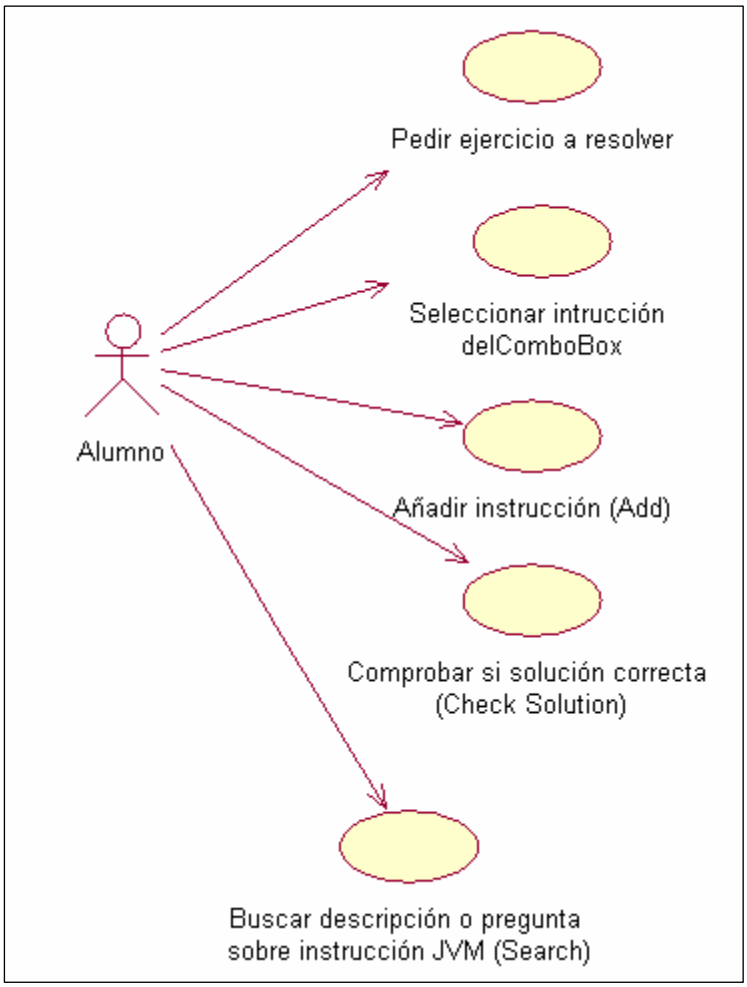


Figura 2.2: Casos de uso del alumno después de cargar el perfil.

o Caso de uso "Pedir ejercicio a resolver":

- Objetivo: Que la aplicación proponga un ejercicio a resolver que se ajuste al perfil del usuario y que no ha resuelto anteriormente.
- Entradas: Ninguna.
- Precondiciones: Se debe haber iniciado el módulo de aprendizaje y cargado o creado un perfil de usuario.

- Salida: Se carga un ejercicio en la ventana de usuario y se propone para que sea resuelto.

- Post-condiciones:

  - Condiciones de final exitoso: Se carga un ejercicio en la ventana de usuario y se propone para que sea resuelto.

  - Condiciones de final fallido: No se carga un ejercicio en la ventana de usuario y se propone para que sea resuelto y se deja al usuario que elija por si mismo.

- Actor primario: Alumno.

o Caso de uso "Selección instrucción del ComboBox[1] o lista desplegable":

- Objetivo: Poder elegir una instrucción de la JVM de un combobox para poder añadir parámetros.

- Entradas: Selección con el ratón de una instrucción de la JVM.

- Precondiciones: Se debe haber iniciado el módulo de aprendizaje y cargado o creado un usuario, y elegido un ejercicio.

- Salida: Se muestra la instrucción seleccionada para que el usuario pueda añadir parámetros.

- Post-condiciones:

- Condiciones de final exitoso: Se muestra la instrucción seleccionada para que el usuario pueda añadir parámetros.

- Actor primario: Alumno.

[1] Véase Glosario de términos

- Caso de uso “Pedir ejercicio a resolver”:

  - Objetivo: Añadir una instrucción a la solución del ejercicio. Es decir seguir resolviendo el ejercicio.

  - Entradas: Instrucción de la JVM.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje y cargado o creado un usuario, seleccionado una.

  - Salida: Añadir una instrucción a la solución del ejercicio.

  - Post-condiciones:

  - Condiciones de final exitoso: Añadir una instrucción a la solución del ejercicio.

  - Actor primario: Alumno.

- Caso de uso “Comprobar si solución correcta”:

  - Objetivo: Intentar comprobar si el ejercicio se ha solucionado correctamente.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje y cargado o creado un usuario.

  - Salida: Se informa si se ha resuelto correctamente un ejercicio, o en caso contrario se nos muestra la ayuda correspondiente al primer error que se ha cometido.

- Post-condiciones:
    - Condiciones de final exitoso: Se informa si se ha resuelto correctamente un ejercicio, o en caso contrario se nos muestra la ayuda correspondiente al primer error que se ha cometido.
- Actor primario: Alumno.

- Caso de uso “Buscar instrucción o pregunta sobre instrucción de la JVM”:
    - Objetivo: Obtener información sobre las instrucciones de la JVM.
    - Entradas: Texto en inglés en forma de descripción o pregunta sobre una instrucción de la JVM.
    - Precondiciones: Se debe haber iniciado el módulo de aprendizaje y cargado o creado un usuario.
    - Salida: Se muestra la información sobre la instrucción que más encaja en la descripción proporcionada.
    - Post-condiciones:
        - Condiciones de final exitoso: Se muestra la información sobre la instrucción que más encaja en la descripción proporcionada.
    - Actor primario: Alumno.

- **Casos de uso pulsando el botón derecho del ratón sobre el código compilado**.

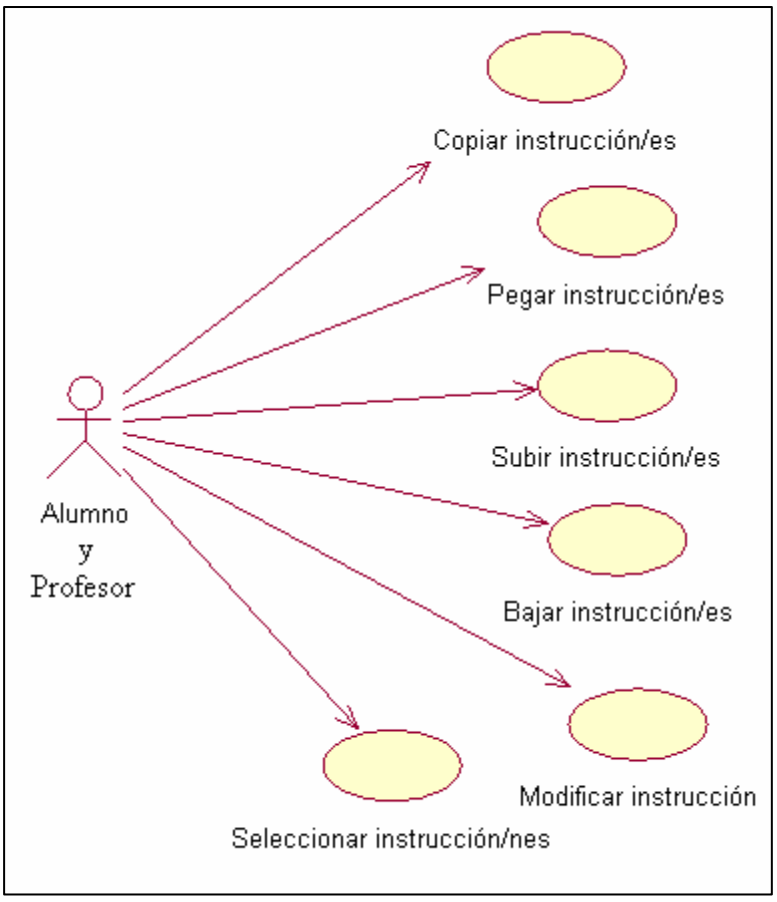


Figura 2.3: Casos de uso del alumno

o Caso de uso "Seleccionar instrucción/es":

  - Objetivo: Que una o varias instrucciones de la JVM introducidas por el usuario como solución al ejercicio se seleccionen.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje, cargado o creado un usuario e introducido una o varias instrucciones de la JVM como solución al ejercicio.

- Salida: Se seleccionan una o varias instrucciones, se quedan marcadas en color azul.

- Post-condiciones:

    - Condiciones de final exitoso: Se seleccionan una o varias instrucciones, se quedan marcadas en color azul.

- Actor primario: Alumno y Profesor.

o Caso de uso “Copiar instrucción/es”:

- Objetivo: Que una o varias instrucciones de la JVM seleccionadas por el usuario se copien en el portapapeles.

- Entradas: Ninguna.

- Precondiciones: Se debe haber iniciado el módulo de aprendizaje, cargado o creado un usuario, introducido una o varias instrucciones de la JVM como solución al ejercicio y hayan sido seleccionadas.

- Salida: Se copian una o varias instrucciones en el portapapeles.

- Post-condiciones:

    - Condiciones de final exitoso: Se copian una o varias instrucciones en el portapapeles.

- Actor primario: Alumno y Profesor.

- Caso de uso “Pegar instrucción/es”:

  - Objetivo: Que una o varias instrucciones de la JVM anteriormente copiadas por el usuario se peguen como parte de la solución.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje, cargado o creado un usuario, introducido una o varias instrucciones de la JVM como solución al ejercicio que hayan sido seleccionadas y copiadas.

  - Salida: Se pegan una o varias instrucciones anteriormente copiadas en el espacio dedicado para componer la solución de código copilado.

  - Post-condiciones:

    - Condiciones de final exitoso: Se pegan una o varias instrucciones anteriormente copiadas en el espacio dedicado para componer la solución de código copilado.

  - Actor primario: Alumno y Profesor.

- Caso de uso “Subir o bajar instrucción/es”:

  - Objetivo: Que una o varias instrucciones de la JVM seleccionadas por el usuario se suba o bajen una posición respecto al orden en el que están escritas.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje, cargado o creado un usuario, introducido una o varias instrucciones de la

JVM como solución al ejercicio y hayan sido seleccionadas. Si se sube la instrucción no puede ser la primera y si se baja no puede ser la última.

- Salida: Se suben o bajan una o varias instrucciones en el portapapeles.

- Post-condiciones:

  - Condiciones de final exitoso: Se suben o bajan una o varias instrucciones en el portapapeles.

- Actor primario: Alumno y Profesor.

- Caso de uso “Seleccionar instrucción/es”:

  - Objetivo: Modificar una instrucción, cambiando el nombre o alguno de los parámetros que la componen.

  - Entradas: Nuevo texto resultado de la modificación de la instrucción.

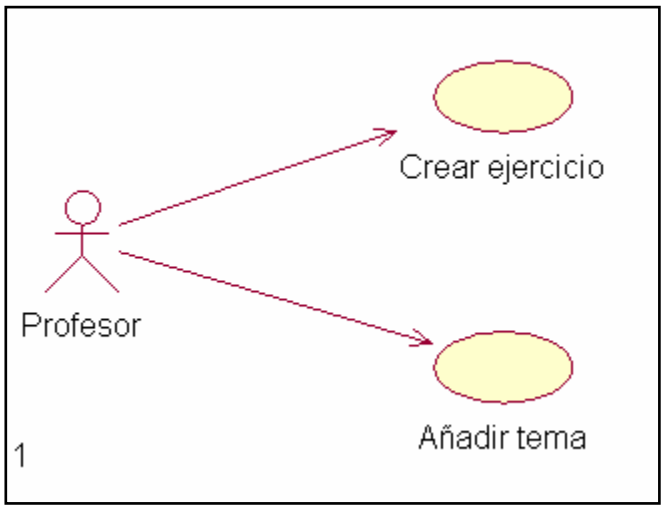
  - Precondiciones: Se debe haber iniciado el módulo de aprendizaje, cargado o creado un usuario e introducido una o varias instrucciones de la JVM como solución al ejercicio.

  - Salida: La instrucción queda modificada.

  - Actor primario: Alumno y Profesor.

2.3.2 Casos de uso del profesor.

- Casos de uso con el editor de ejercicio.



o Caso de uso “Crear ejercicio”:

- Objetivo: Creación del ejercicio escribiendo el código, el título, el nivel y al descripción.
- Entradas: Código java, titulo, nivel y descripción.
- Precondiciones: Se debe haber iniciado el editor de ejercicio.
- Salida: Ninguna.
- Actor primario: Profesor.

o Caso de uso “Añadir tema”:

- Objetivo: Añadir un tema, que expone el tipo de instrucciones utilizadas en el ejercicio.
- Entradas: La selección de un tema de la lista de temas de la JVM.

- Precondiciones: Se debe haber iniciado el editor de ejercicios y haber seleccionado un tema de la lista de temas.

- Salida: Aparece en la lista de temas, el nuevo tema seleccionado.

- Actor primario: Profesor.

- **Casos de uso al pulsar botón "Actions"**

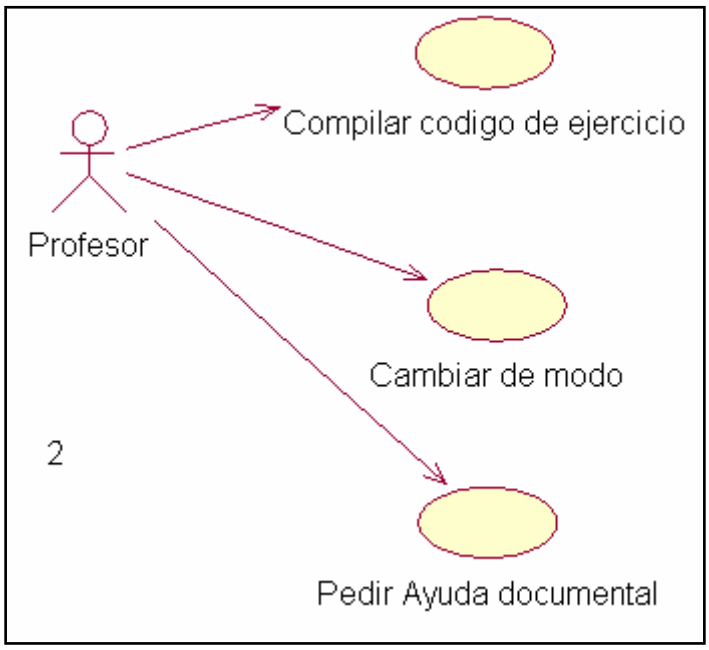


Figura 2.5: Casos de uso de las acciones del profesor.

o Caso de uso "Compilar código de ejercicio":

- Objetivo: Comprobar si el código Java introducido por el profesor o cargado es correcto, es decir, si compila correctamente.

- Entradas: Ninguna.

- Precondiciones: Se debe haber iniciado el editor de ejercicios y haber cargado o escrito un ejercicio.

- Salida: Aparece una ventan que muestra el resultado de la compilación.

- Post-condiciones:

    - Condiciones de final exitoso: Aparece una ventan que muestra que el resultado de la compilación es correcto.

    - Condiciones de final erróneo: Aparece una ventana que muestra que el resultado de la compilación es erróneo.

- Actor primario: Profesor.

o Caso de uso “Cambiar Modo”:

- Objetivo: Cambiar del modo editor de ejercicios (sólo se puede escribir el código java del ejercicio el nivel y los temas del ejercicio) al editor de casos (más opciones que el editor de ejercicios, creación de ayudas y código erróneo) o viceversa, es decir, cambiar del modo editor de casos al modo editor de ejercicios.

- Entradas: Ninguna.

- Precondiciones: Se debe haber iniciado el editor de ejercicios.

- Salida: Se cambia de modo. En el cambio de modo no se pierde ninguna configuración realizada, es decir, se guardan las ayudas, errores, temas, código correcto, código Java, etc.

- Post-condiciones:

    - Condiciones de final exitoso: Aparece una ventan que muestra que el resultado de la compilación es correcto.

    - Condiciones de final erróneo: Aparece una ventan que muestra que el resultado de la compilación es erróneo.

- Actor primario: Profesor.

o Caso de uso “Pedir Ayuda documental”:

- Objetivo: Mostrar al usuario la ayuda de la aplicación y del libro de la máquina virtual de Java.

- Entradas: Ninguna.

- Precondiciones: Se debe haber iniciado el módulo de aprendizaje.

- Salida: Se muestra el libro de la JVM en el explorador de Internet por defecto del computador del usuario.

- Post-condiciones:

    - Condiciones de final exitoso: Se abre el explorador de Internet y se muestra el libro en HTML de la JVM.

    - Condiciones de final fallido: Si no esta bien configurado la aplicación encargada de abrir los archivos HTML en el computador del usuario, podría abrirse con otra aplicación en formato texto , o pedir que se seleccione una de una lista de aplicaciones.

- Actor primario: Profesor.

- **Casos de uso pulsando botón "File"**

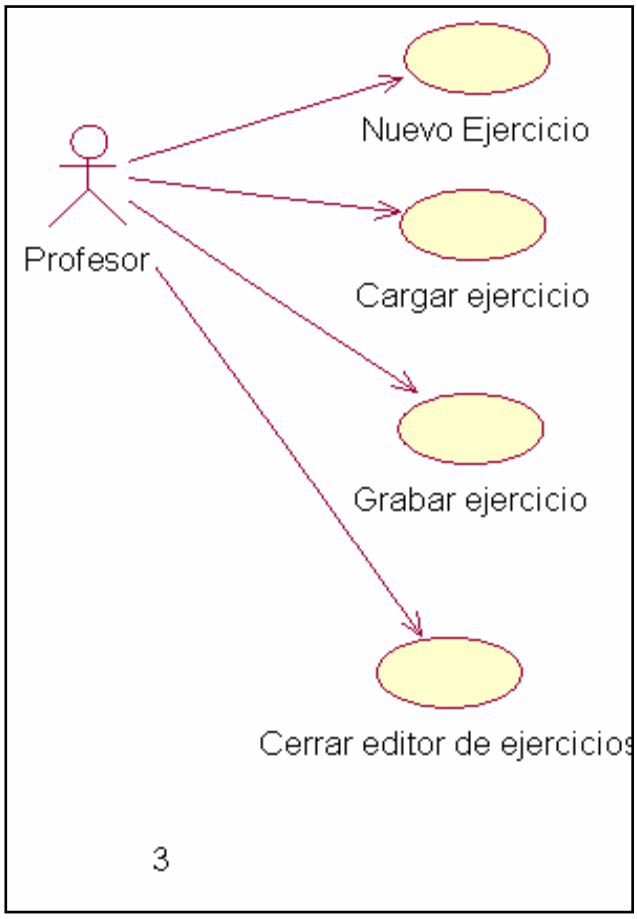


Figura 2.6: Casos de uso del profesor sobre ficheros.

o Caso de uso "Nuevo Ejercicio":

- Objetivo: Se reinician todas las ventanas y se ponen en blanco para poder crear un nuevo ejercicio.

- Entradas: Ninguna.

- Precondiciones: Se debe haber iniciado el editor de ejercicios.

- Salida: Se reinician todas las ventanas y se ponen en blanco para poder crear un nuevo ejercicio

- Actor primario: Profesor.

- Caso de uso “Cargar ejercicio”:

  - Objetivo: Carga un ejercicio previamente grabado si esta en modo ejercicio o carga una ventana donde elegir un caso si esta en el editor de casos.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el editor de ejercicios, da igual en que modo.

  - Salida: Carga un ejercicio previamente grabado si esta en modo ejercicio o carga una ventana donde elegir un caso si estamos en el editor de ejercicios.

  - Actor primario: Profesor.

- Caso de uso “Grabar un ejercicio”:

  - Objetivo: Graba un ejercicio previamente cargado si esta en modo ejercicio o graba caso si esta en el editor de ejercicios.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el editor de ejercicios, da igual en que modo. Además se debe haber rellenado todos los campos de ambos modos y el ejercicio tiene que haber compilado correctamente.

- Salida: Graba un ejercicio previamente cargado si esta en modo ejercicio o graba caso si esta en el editor de ejercicios.

- Actor primario: Profesor.

o Caso de uso “Cerrar ejercicio”:

- Objetivo: Cierra el editor de ejercicios.

- Entradas: Ninguna.

- Precondiciones: Se debe haber iniciado el editor de ejercicios, da igual en que modo.

- Salida: Cierra el editor de ejercicios.

- Actor primario: Profesor.

- **Casos de uso en el editor de ejercicios en el modo editor de casos.**

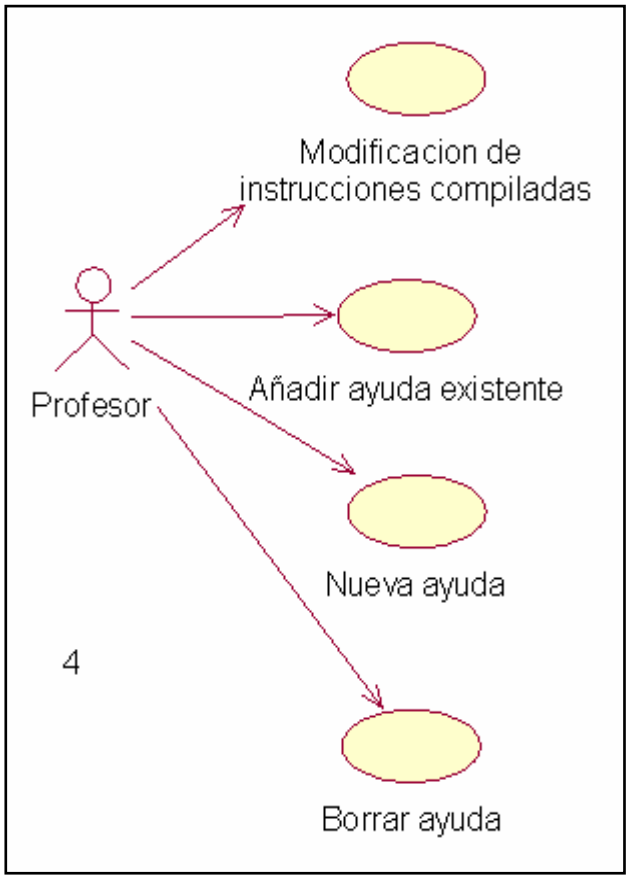


Figura 2.7: Casos de uso del profesor sobre casos.

- Caso de uso "Modificación de instrucciones compiladas":

  - Objetivo: Modificar una de las instrucciones compiladas.

  - Entradas: Texto correspondiente a la modificación.

  - Precondiciones: Se debe haber iniciado el editor de ejercicios, y el modo editor de casos.

  - Salida: La instrucción seleccionada se modifica.

  - Actor primario: Profesor.

- Caso de uso “Añadir ayuda existente”:

  - Objetivo: Añadir una ayuda existente a un caso, seleccionándola de entre todas las guardadas en la base de casos.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el editor de ejercicios, y el modo editor de casos.

  - Salida: La ayuda seleccionada se añade al caso que se esta creando.

  - Actor primario: Profesor.

- Caso de uso “Nueva ayuda”:

  - Objetivo: Añadir una nueva ayuda, escribiéndola en el lugar correspondiente.

  - Entradas: El texto de la nueva ayuda.

  - Precondiciones: Se debe haber iniciado el editor de ejercicios, y el modo editor de casos.

  - Salida: La nueva ayuda es creada y asignada al caso que se está creando.

  - Actor primario: Profesor.

- Caso de uso “Borrar ayuda”:

  - Objetivo: Borra una ayuda que se ha añadido o creado.

  - Entradas: Ninguna.

  - Precondiciones: Se debe haber iniciado el editor de ejercicios, seleccionado el modo editor de casos y añadido o creado al menos una ayuda. Además se ha tenido que seleccionar una de las ayudas creadas.

  - Salida: La ayuda seleccionada se borra del caso que se está editando.

  - Actor primario: Profesor.

2.4 DISEÑO MODULAR

Se basa en la idea de descomponer un problema en subproblemas más sencillos. Hacer una aplicación modular aporta varias ventajas, tanto para el desarrollo de la aplicación como para modificar el producto una vez terminado.

El objetivo es definir las funciones principales de cada módulo, sin entrar en muchos detalles. La definición detallada de los módulos queda definida en el capítulo 5 de este documento.

El sistema posee los siguientes módulos:

- Gestor de ayudas.

  Es el encargado de aportar una ayuda útil al usuario, es decir, que le permita continuar resolviendo el ejercicio en caso de que no sepa cómo hacerlo. En este módulo se abordarán técnicas de inteligencia artificial.

- Módulo de aprendizaje del alumno:

  Su principal función es la interactuación con el alumno.

- Editor de ejercicios y casos:

  Este módulo permite interactuar con el profesor, y aporta facilidades para la creación de ejercicios.

- Información del sistema:

  Almacena internamente toda la información que necesite la aplicación. Es la manera de acceder al conocimiento exterior por parte del sistema, así pues el resto de módulos solicitan la información que precisen a este módulo. Permite separar la información de la gestión que se haga con dicha información.

  A continuación mostramos gráficamente cómo interactúan entre si: **Figura 2.8**

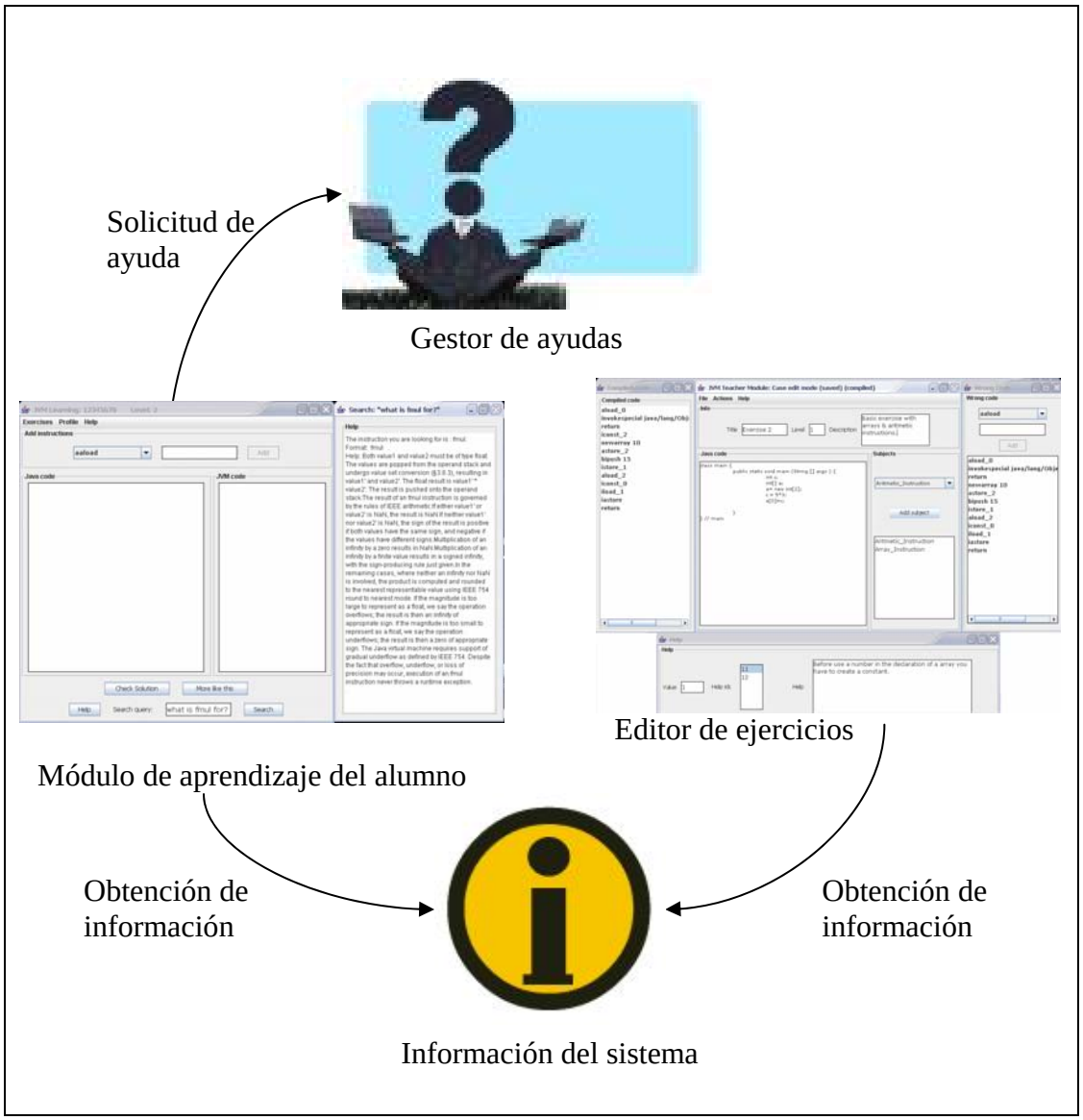


Figura 2.8: Diagrama de comunicación entre módulos

CAPÍTULO 3
TECNOLOGÍAS UTILIZADAS

3. TECNOLOGIAS UTILIZADAS

3.1 INTRODUCCIÓN

Antes de afrontar cualquier tipo de proyecto es necesario documentarse he investigar con el fin de identificar qué tipo de herramientas son útiles para la elaboración de la aplicación, así como el estudio de aquellas que se decidan usar.

El estudio llevado a cabo en esta fase se centra principalmente en cómo representar el conocimiento del dominio que se desea enseñar, y en cómo dar inteligencia al módulo gestor de ayudas, para que guíe de manera inteligente al alumno para resolver un ejercicio.

3.2 ONTOLOGÍA

Puesto que uno de los requisitos es separar el conocimiento del razonamiento, surge el problema de cómo implementar el conocimiento sobre el que se va a enseñar a los alumnos.

En esta fase tomamos la decisión de implementar el conocimiento en formato OWL, que describimos a continuación, mediante una herramienta de edición de bases de conocimiento llamada Protégé.

3.2.1 Ontology Web Language (OWL).

Ha sido desarrollado por el W3C y tiene como punto de partida las experiencias previas realizadas con DAML-OIL en las cuales se inspiraron los creadores de OWL para desarrollar el lenguaje.

Es un lenguaje de marcado para la publicación de ontologías en Internet y tiene como objetivo facilitar un modelo de marcado, construido sobre RDF y codificado en XML que permita representar ontologías a partir de un vocabulario más amplio y una sintaxis mas fuerte que la que permite RDF. Por

este motivo OWL puede ser utilizado para representar de forma explicita términos pertenecientes a un dominio de conocimiento, en nuestro caso las instrucciones de la maquina virtual de Java y definir las relaciones que existen entre ellas.

OWL se divide en tres sublenguajes OWL-Lite, OWL-DL y OWL-Full, cada uno de los cuales proporciona un conjunto definido sobre el que trabajar, siendo el mas sencillo OWL-Lite y el más completo OWL-Full.

```
<?xml version="1.0" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:owl="http://www.w3.org/2002/07/
    ontologies.com/unnamed.owl#" xml:base="http://www.owl-ontologies.com/unnamed.owl">
    <owl:Ontology rdf:about="" />
  - <owl:Class rdf:ID="Ldc_Instruction">
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="Instruction" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:ID="Add_Instruction">
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="Aritmetic_Instruction" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:ID="ireturn">
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="Return_Instruction" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:ID="fsub">
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="Sub_Instruction" />
      </rdfs:subClassOf>
    - <rdfs:subClassOf>
      - <owl:Restriction>
        - <owl:onProperty>
            <owl:ObjectProperty rdf:ID="has_parameter" />
          </owl:onProperty>
        - <owl:someValuesFrom>
            <owl:Class rdf:ID="float" />
          </owl:someValuesFrom>
```

Figura 3.1: Representación XML de la ontología.

Para la realización de la ontología en OWL hemos empleado el editor de base de conocimientos Protégé.**[Protégé]**

3.2.2 Protégé

Es una aplicación desarrollada por el Stanford Medical Informatics en la Stanford University School of Medicine con apoyo de instituciones militares, medicas, tecnológicas y científicas de los Estados Unidos. Es un editor de base de conocimientos de código abierto y de libre distribución.

Posee un editor gráfico para la creación de conocimiento en distintos formatos como Java, HTML, Clips, Owl, RDF, etc. Así como métodos para la transformación del conocimiento de un formato a otro.

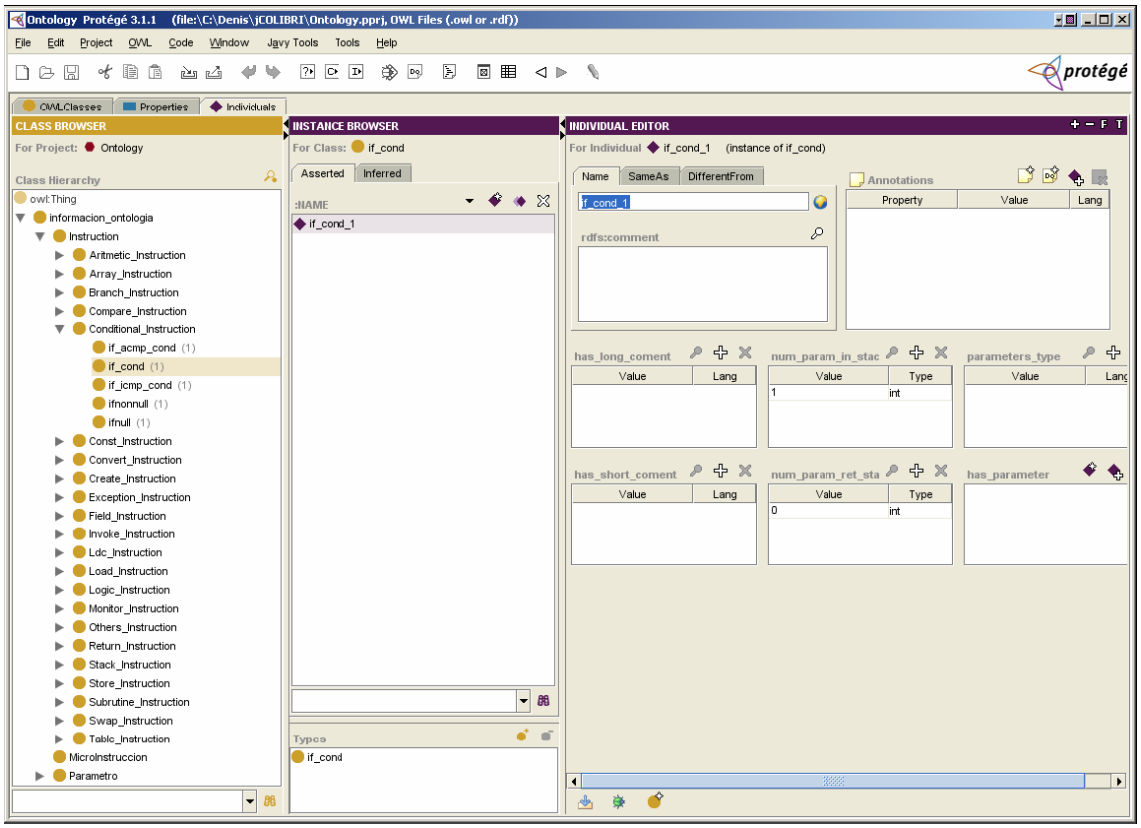


Figura 3.2: Ontología interpretada por Protégé

3.2.3 Jena.

Es un sistema de código abierto que permite la conexión con diferentes tipos de razonadores. Jena es un framework de Java para la construcción de aplicaciones de Web semántica. Proporciona un entorno de programación para

RDF, RDFS, SPARQL y OWL e incluye un motor de inferencia basado en reglas. Jena aporta los métodos necesarios para la conexión Java Owl.

3.2.4 Conector Java-OWL

Ante la necesidad de conectar de forma sencilla el sistema de enseñanza con la ontología, desarrollamos un conector Java-Owl. Este conector implementa los métodos necesarios para el acceso a cualquier ontología en formato OWL.

Posee métodos para crear y modificar fichero OWL, la necesidad surge del problema a la hora de implementar la ontología, que se detalla en el **capítulo 4.3.1.**

Su funcionamiento viene definido por los distintos interfaces:

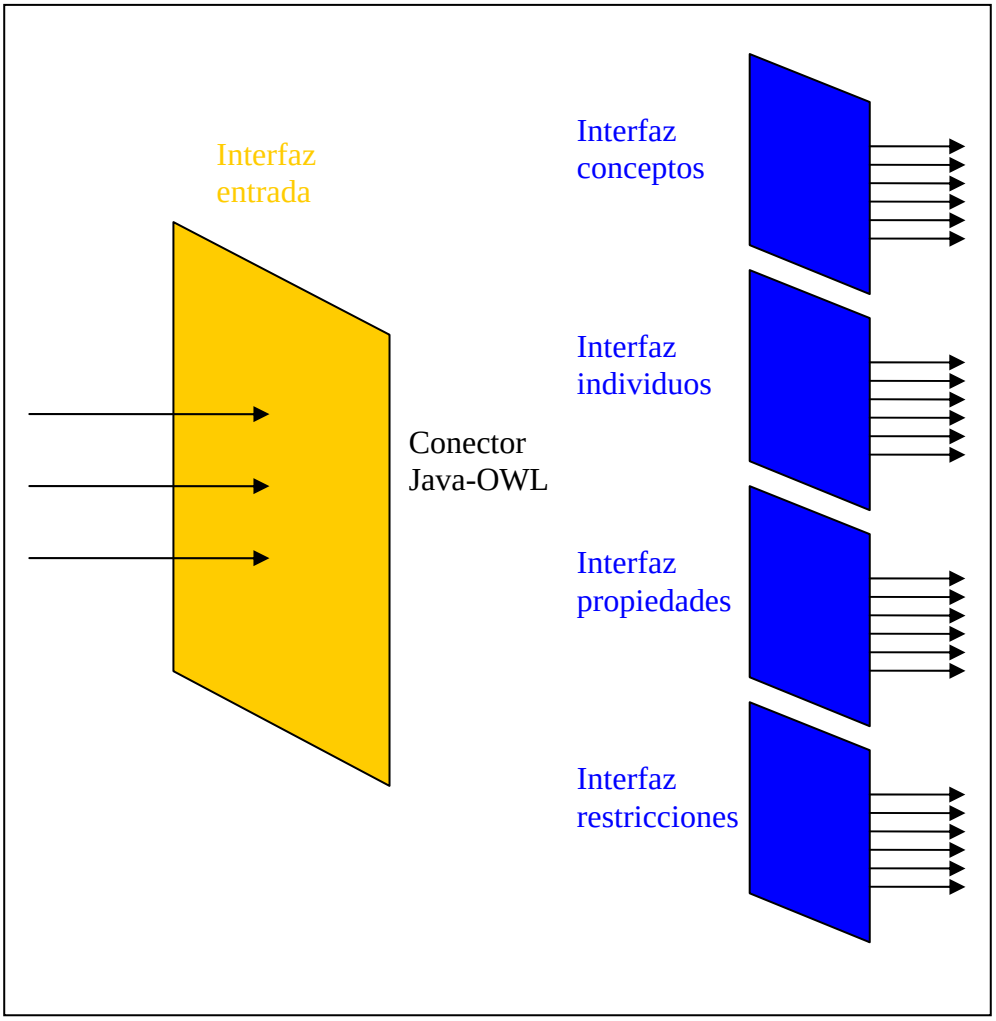


Figura 3.3: Diseño analítico del conector Java-Owl.

La interfaz de entrada posee métodos para elegir y traducir un fichero OWL.

El acceso a la información de la ontología se hace mediante los métodos que proporcionan los interfaces de salida. Según a que tipo de conocimiento se quiera acceder se usarán los métodos de uno u otro interfaz.

3.3 RAZONAMIENTO BASADO EN CASOS

3.3.1 Introducción

Nos planteamos que tecnología sería la más apropiada para la creación de un sistema de enseñanza interactiva. Ya que necesitábamos un sistema donde el aprendizaje jugara un papel preponderante, con la experiencia y ayuda de GAIA (Group for Artificial Intelligence Applications) perteneciente al departamento de sistemas informáticos y programación decidimos decantarnos por la creación de un sistema de razonamiento basado en casos.

3.3.2 Descripción de un sistema CBR (Case Based Reasoning)

El CBR es un modo natural de razonamiento de los seres humanos, es decir, los seres humanos nos basamos en experiencias pasadas para encontrar solución a los problemas que se nos plantean. Cuando una persona se enfrenta a un problema busca consciente o inconscientemente una solución que funcionó en el pasado, o sino encontramos una experiencia vivida igual buscamos una experiencia similar y adaptamos nuestra actuación para la situación actual. Este es exactamente el funcionamiento de un sistema de razonamiento basado en casos. Una cita que refleja lo anteriormente expuesto es:

"Un razonador basado en casos resuelve nuevos problemas adaptando las soluciones que fueron utilizadas para resolver problemas previos similares"
[Riesbeck and Schank, 1989]

Las partes más importantes de un sistema CBR son:

- **Caso**: Un caso contiene la descripción del problema y una o varias soluciones a este problema.

  "Un caso es un fragmento contextualizado de conocimiento que representa una experiencia y que enseña una lección importante para conseguir los objetivos del razonador" **[Kolodner & Leake 1997]**

- **Base de casos**: es una librería de casos pasados, es decir es donde se van almacenando los casos anteriores y serán estos los que nos servirán para razonar y encontrar la solución al problema a resolver.

- Función de similitud: esta función se encarga de computar la similitud entre dos casos. Se utiliza para comparar el caso actual con los casos almacenados en la base de casos, su cometido es decidir cuan parecidos son dos casos entre si.

- Revisión y adaptación: si la solución encontrada no solventa de manera óptima el problema, se revisa y adapta para mejorarla.

El funcionamiento de un sistema CBR es el siguiente: Para solucionar un problema concreto se buscan casos similares en la base de casos. Los casos recuperados se utilizan para sugerir una solución, la cual se usa para intentar solucionar el problema. Sí fuera necesario la solución se modifica y revisa. Finalmente se guarda el problema actual junto a la solución como un nuevo caso.

3.3.3 Razonamiento basado en casos textuales

3.3.3.1 Introducción

Ante la necesidad de ayudar al alumno a resolver los ejercicios nos planteamos cómo prestarle una ayuda fácil e intuitiva a cerca de las

instrucciones de la JVM, de esta manera el alumno no tendría que saber de memoria todas y cada una de las instrucciones, o incluso podría describir el funcionamiento de una instrucción y recibir como resultado la instrucción que tiene esa funcionalidad.

Por este motivo nos planteamos la posibilidad de incorporar a la aplicación un sistema de ayuda textual capaz de procesar preguntas en inglés y dar una respuesta con la información solicitada. Es decir, el alumno introduce una descripción de una instrucción o una pregunta concreta en inglés y nuestro sistema le dice qué instrucción de la JVM es la que esta buscando.

3.3.3.2 Descripción de un sistema CBR textual.

El CBR textual es una subdisciplina cada vez más importante del CBR. Las técnicas de CBR textual facilitan la construcción de sistemas CBR reduciendo o eliminando las tareas de caracterización de casos difíciles en los que el texto esta semi-estructurado o no estructurado.

El funcionamiento del CBR textual es muy parecido a un sistema CBR estándar, la diferencia radica en la construcción de los casos y las funciones de similitud que se aplicarán. En un sistema CBR textual no se pueden almacenar los textos completos sin estructurarlos e indexarlos, ni suponer una búsqueda secuencial, es decir lectura línea a línea de dicho texto, porque este proceso implicaría una gran perdida de eficacia y un aumento considerable tanto del tiempo como de la complejidad de una consulta.

Por ello el CBR textual construye estructuras de datos sobre el texto, asignando índices para acelerar las búsquedas. Estas estructuras de datos conforman claves que identifican las palabras, frases, etc. contenidas en el texto de la consulta.

El proceso de creación de estas estructuras suele ser el mismo para los distintos métodos: En primer lugar se quitan las palabras vacías, es decir

carentes de significado (determinantes, preposiciones, conjunciones…). A continuación se extraen las raíces de los verbos y las palabras que conforman el texto, eliminando los prefijos, sufijos, etc. Después se analizan la frecuencia de aparición de cada uno de los términos extraídos. Y por último se guarda una lista de términos para poder aplicar el modelo de espacio vectorial en el cómputo de la similitud.

Gracias a la creación de estas estructuras, el proceso de recuperación se acelera enormemente, ya que bastan con acceder a los documentos dada una lista de palabras clave de la consulta.

En resumen el sistema CBR textual es óptimo para a partir de una pegunta o una descripción de una instrucción de la JVM buscar en la documentación y devolver una respuesta acertada.

3.3.3.3 JColibri.

Para la creación del sistema CBR textual se nos facilito una herramienta desarrollada por GAIA llamada jCOLIBRI. **[JColibri].**

JColibri es framework orientado a objetos en Java para la creación de aplicaciones de razonamiento basado en casos (sistemas CBR).

"JColibri ha sido diseñado pensando en conseguir una plataforma de desarrollo de sistemas CBR que sirva de referencia en la comunidad científica. El objetivo es conseguir que el framework crezca y evolucione gracias a las aportaciones de distintos especialistas.
jCOLIBRI ha sido construido en torno a la idea básica de separar tareas (tasks) y métodos (methods). Las tareas indican objetivos que el sistema debe alcanzar y guían la ejecución de la aplicación. Los métodos indican como resolver las tareas. Por ejemplo, una idea bastante aceptada es que el ciclo principal de CBR puede descomponerse en cuatro tareas: recuperar los casos mas similares (retrieve), reutilizarlos para resolver el problema

(resuse), revisar la solución propuesta (revise) y aprender de la experiencia (retein). Siguiendo esta idea, jCOLIBRI modela el ciclo CBR mediante la tarea CBR Task y propone un método asociado CBR Method que la descompone en 4 subtareas: CBR Retrive Task, CBR Reuse Task, CBR Revise Task y CBR Retein Task. A su vez para resolver estas tareas habrá que seleccionar los métodos correspondientes."[2]

[2] http://gaia.fdi.ucm.es/people/pedro/aad/antonio_sanchez.pdf

CAPÍTULO 4
FASE DE ADQUISICIÓN DE CONOCIMIENTO

4. FASE DE ADQUISICIÓN DE CONOCIMIENTO

4.1 INTRODUCCIÓN

En esta fase abordamos uno de los principales problemas de los sistemas basados en conocimiento, que es la adquisición del mismo. La adquisición de conocimiento se compone de dos etapas:

- Creación de los casos y la base de casos de los dos sistemas CBR:
    - Sistema CBR de ayuda textual: Para la creación de la base de casos decidimos definir un dominio de conocimiento sobre el que va a enseñar la aplicación, la JVM. Optando por emplear técnicas de extracción automática de conocimiento y creación automática de los casos y la base de casos.
    - Sistema CBR de ayuda contextual: como obtener los casos que componen la base de casos.
- Representación del conocimiento en la herramienta a emplear. Este problema se solucionó en la fase de investigación. Optando por representarlo en una ontología en formato OWL.

4.2 CASOS Y BASE DE CASOS

4.2.1 Introducción

En este apartado se describe detalladamente la creación y estructura de las bases de casos y los casos que los compone (definidos en el **capitulo 3.3.2**), del sistema. Se diferencian dos apartados en primer lugar la base de casos y casos del sistema CBR textual y en segundo lugar del sistema CBR tradicional con la documentación contextual aportada por el profesor.

4.2.2 Base de casos para el sistema CBR Textual.

Uno de nuestros primeros objetivos fue el procesamiento de la documentación de la maquina virtual de Java (JVM) **[JVM]** para convertirla en una base de casos textual y sobre esta base de casos diseñar un sistema CBR textual capaz de contestar preguntas sobre instrucciones de la JVM.

El conocimiento de nuestro sistema CBR Textual ha sido sacado del libro *The Java™ Virtual Machine Specification Copyright © 1999 Sun Microsystems, Inc.* **[JVM]**

El procesado de la documentación, inicialmente en formato HTML, resulto un proceso muy dependiente de la información y el formato especifico del archivo. Además tenía mucha información innecesaria y este formato no era muy apropiado para una fácil recuperación de la información. Por ello decidimos parsear el fichero a un formato predefinido por nosotros para la óptima extracción de información. Así pues mediante un programa en Java se ha parseado a texto plano este libro.

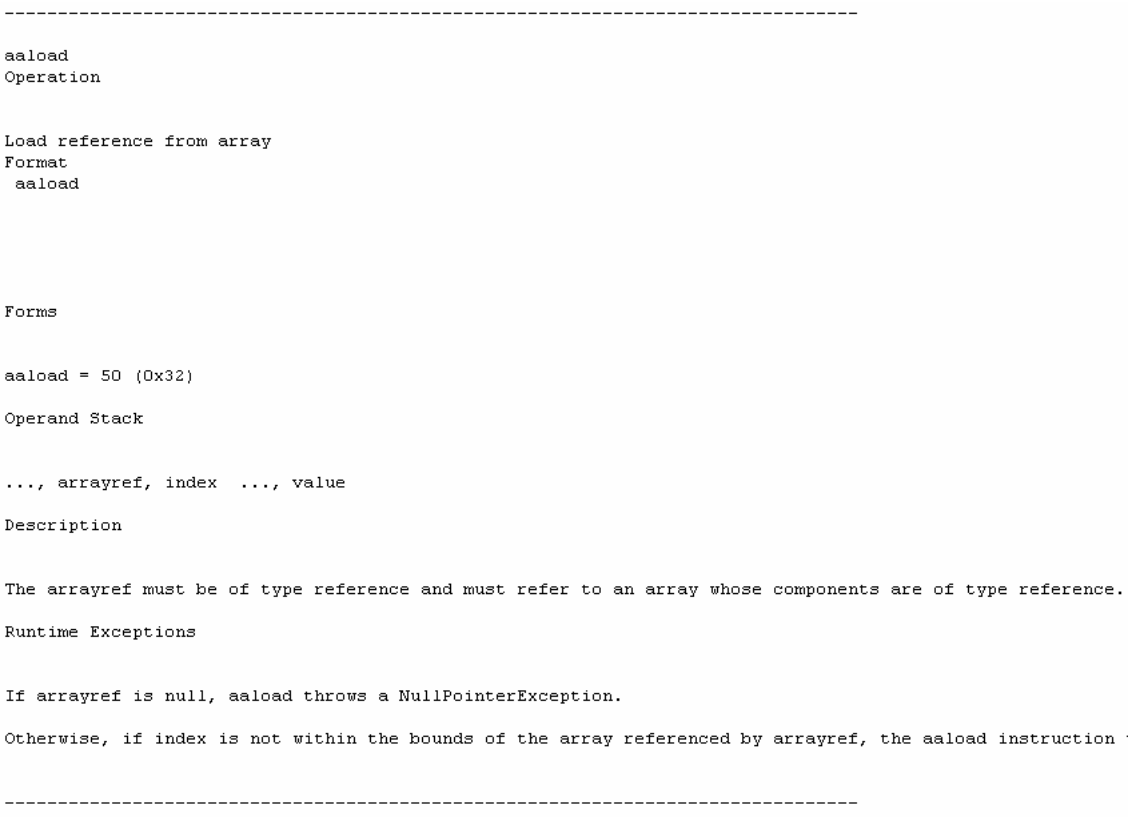
El formato inicial de la documentación consta de 55 paginas HTML, sin ningún tipo de formato ni estructura.

```
<a href="Instructions2.doc11.html">P</a>
<a href="Instructions2.doc12.html">R</a>
<a href="Instructions2.doc13.html">S</a>
<a href="Instructions2.doc14.html">T</a>
<a href="Instructions2.doc15.html">W</a>

<a name="baload"></a>
<hr><h2>baload</h2>
<a name="baload.Operation"></a>
<p><b>Operation</b><br>
<blockquote><a name="67301"></a>
Load <code>byte</code> or <code>boolean</code> from array<p><Table Border="1">
</blockquote>

<p><b>Format</b><br>
<blockquote>

<tr><td><a name="67300"></a>
 <i>baload</i>
<td><a name="87568"></a>


</Table><br></blockquote><p>
<a name="baload.Forms"></a>
<p><b>Forms</b><br>
<blockquote><a name="67302"></a>
<i>baload</i> = 51 (0x33)</blockquote><p>
<a name="baload.Operand"></a>
<p><b>Operand Stack</b><br>
<blockquote><a name="67303"></a>
..., <i>arrayref</i>, <i>index</i> <img src="chars/arrwdbrt.gif"> ..., <i>value</i>
```

Figura 4.1: Formato inicial del libro sobre la máquina virtual de JAVA.

El archivo parseado esta formado por todas las instrucciones JVM cada una de ellas delimitada por una línea de guiones. Además cada instrucción tiene un formato y una estructura fija:

- *Nombre* de la instrucción.
- *Operation*: pequeña descripción de la instrucción.
- *Format*: formato de la instrucción.
- *Forms* : información sobre la codificación de la instrucción.
- *Operand Stack*: operandos que tiene que estar apilados en la pila.
- *Description*: descripción completa de la instrucción.



```
--------------------------------------------------------------------------------

aaload
Operation


Load reference from array
Format
 aaload




Forms


aaload = 50 (0x32)

Operand Stack


..., arrayref, index  ..., value

Description


The arrayref must be of type reference and must refer to an array whose components are of type reference.

Runtime Exceptions


If arrayref is null, aaload throws a NullPointerException.

Otherwise, if index is not within the bounds of the array referenced by arrayref, the aaload instruction

--------------------------------------------------------------------------------
```

Figura 4.2: Estructura final de la documentación de la JVM.

A partir de esta documentación con el nuevo formato se crea la base de casos gracias a la implementación de un conector en JAVA encargado de leer el archivo parseado y crear uno a uno los casos textuales. La base de casos esta formada por cerca de 180 casos.

La estructura de los casos esta guardada en un archivo .xml:

- *Nombre* instrucción.
- *Formato* de instrucción
- *Descripción* de la instrucción.
-

Cada caso tiene asignado una función de similitud y un peso.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Case>
  - <Description globalSim="Average">
      <SimpleAttribute localSim="Equal" name="instruction" type="String" weight="1.0" />
      <SimpleAttribute localSim="Equal" name="format" type="String" weight="1.0" />
      <SimpleAttribute localSim="Cosine Coefficient" name="description" type="Text" weight="1.0" />
    </Description>
    <Solution globalSim="Average" />
    <Result />
  </Case>
```

Figura 4.3: Estructura del archivo XML con la información de los casos CBR textual.

En el apartado de desarrollo (**Capítulo 6.2.2**) detallamos más en profundidad la implementación de las estructuras internas de los casos y la base de casos.

4.2.3 Base de casos para el sistema CBR de ayuda contextualizada.

En esta ocasión no se tiene ningún tipo de información de donde sacar el conocimiento creando a partir de ellos automáticamente los casos y la base de casos. Por ello es necesario que el profesor se encargue de crear los casos.
Un caso consta básicamente de dos partes:

- Ejercicio: propuesto por el profesor, la solución es compilada y generada automáticamente por el módulo de creación de ejercicios. El profesor únicamente aporta un fichero “.java”.

- La parte de ayuda al alumno: En este caso el profesor debe crear ayudas a errores comunes del ejercicio propuesto.

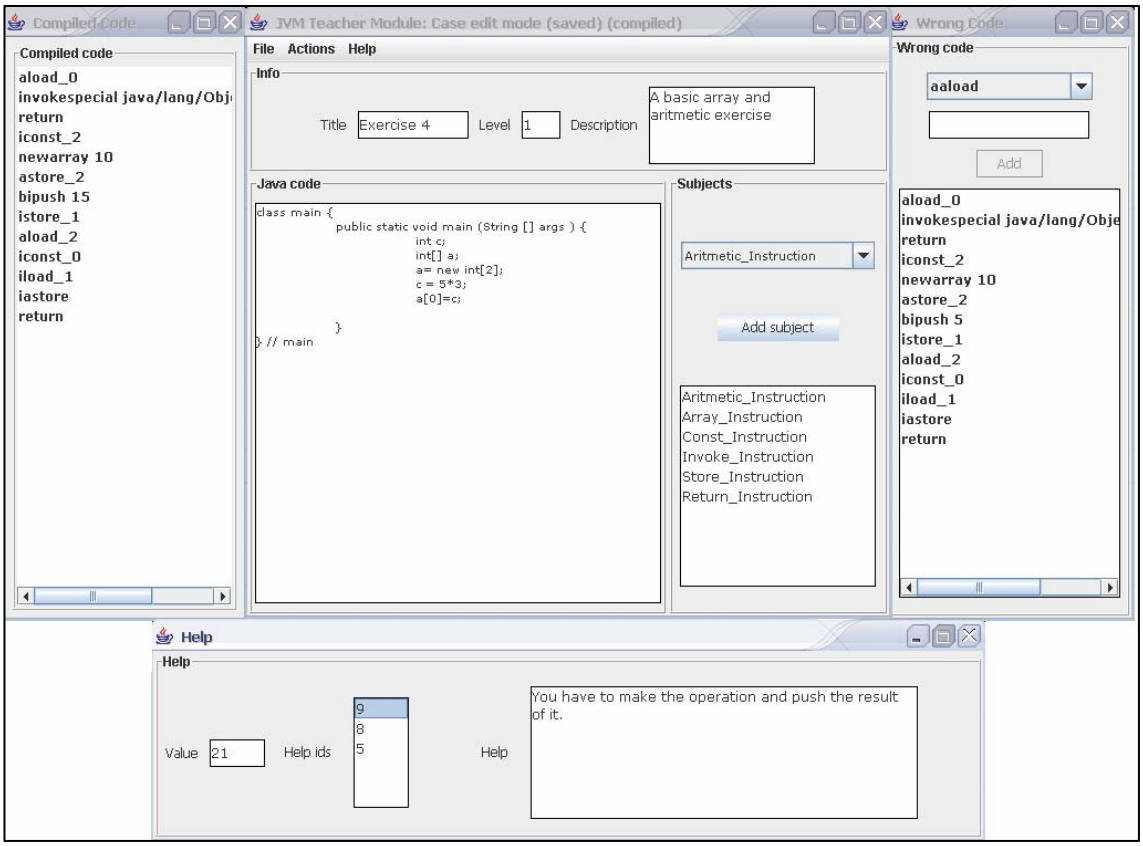


Figura 4.4: Editor de casos y ejercicios.

Con un intuitivo interfaz gráfico el profesor crea un caso CBR y simplemente pulsando “Save” (**Capítulo 6.5**) el caso queda almacenado en la base de casos.

En el apartado de desarrollo (**Capítulo 6.3**) detallamos más en profundidad la implementación de las estructuras internas de los casos y la base de casos.

4.3 JAVA VIRTUAL MACHINE

4.3.1 Creación de la ontología de la JVM.

El problema está en que hacer una ontología para las 148 instrucciones de las que se compone la JVM, es una tarea tediosa, por lo que el proceso de creación de la ontología se hizo de forma automática.

Para conseguir este objetivo se emplearon los métodos de creación y modificación de ontologías OWL que ofrece el conector Java-Owl.
Se añadieron los conceptos y relaciones que representan el conocimiento de la JVM, el resultado se detalla en el siguiente apartado.

4.3.2 Ontología de la JVM.

Representa el conjunto de instrucciones de la máquina virtual de java. Como en una arquitectura de un computador, el conjunto de instrucciones de la JVM indica al procesador cómo realizar una acción.
Hemos definido 148 instrucciones que se relacionan en la Ontología según diferentes aspectos que pueden ser interesantes a la hora de querer utilizar dichas instrucciones.

Los conceptos representan dos aspectos, por un lado las 148 instrucciones de la JVM, y por otro los temas de los que trata cada una. Cada instrucción tiene al menos un tema asociado. Algunos de estos temas son: Instrucción aritmética, lógica, de Salto, de Load y de Store. La característica del tema de la instrucción nos parece muy relevante, por ello hemos incluido en la jerarquía de temas algunos subtemas, como por ejemplo desplazamientos aritméticos, desplazamientos lógicos o instrucciones de tablas. Esta información está organizada en forma de herencia, es decir aquella cuyo tema sea de Salto, heradará de Branch Instruction.

Las relaciones introducidas no son muchas, puesto que para una ontología de la JVM no ha sido fácil extraer todo el conocimiento necesario para introducir más cantidad de relaciones. Una relación se define por su rango y su dominio, el dominio indica el sujeto de la relación, y el rango son los valores de la relación, por ejemplo, para la relación tine_hijo, el domino sería padre, y el rango sería por ejemplo persona, de esta forma decimos que cualquiera que sea padre tiene un hijo que es persona.

Las **relaciones** de la JVM se detallan a continuación definiendo para cada una su rango y su dominio:

- has_parameter_in_stack -> Relaciona 2 conceptos. El dominio de la relación es 'Instruction' y el rango es Parámetro, esto es porque una instrucción posee un tipo de parámetro. Mediante esta relación podremos definir el tipo de parámetros que usa cada instrucción, es decir, el tipo de los elementos que tienen que estar en la pila de operandos antes de invocar la instrucción para que esta se ejecute correctamente.

- has_parameter_type -> Igual que la anterior, pero en este caso los parámetros no hacen referencia a la pila. Algunas instrucciones reciben parámetros explícitos, que no están almacenados en la pila, y esta relación sirve para definir ese tipo de parámetros.

- has_short_coment -> Almacena una breve descripción de lo que hace la instrucción. En este caso el rango es una cadena de caracteres (String) y el dominio 'Instrucion'.

- has_long_coment -> El rango y el domino son los mismos que en has_short_coment. En esta propiedad se almacena una descripción detallada de lo que hace la instrucción, si se deseasen crear individuos concretos (instrucciones particulares de un determinado programa) se puede introducir en este campo todas peculiaridades que pueda tener esa instrucción, como por ejemplo un tipo particular de parámetros que no estuviera definido en la ontología.

- Has_param_ret_stack -> El domino es ‘Instruction’y el rango es Int. Indica el número de parámetros que devuelve la instrucción.

- num_param_in_stack -> El domino es ‘Instruction’y el rango es Int. Indica el número de parámetros que tiene que haber en la pila antes de invocar la instrucción.

- num_param_ret_stack -> El domino es ‘Instruction’y el rango es Int. Indica el número de parámetros que deja en la pila tras la ejecución de la instrucción.

- return_parameter_in_stack -> El domino es ‘Instruction’y el rango es ‘Parameter’. Identifica el tipo de los parámetros que necesita que haya en la pila antes de la ejecución de la instrucción.

- return_parameter_type -> El domino es ‘Instruction’y el rango es ‘Parameter’. Indica el tipo de los parámetros que deja en la pila tras la ejecución de la instrucción.

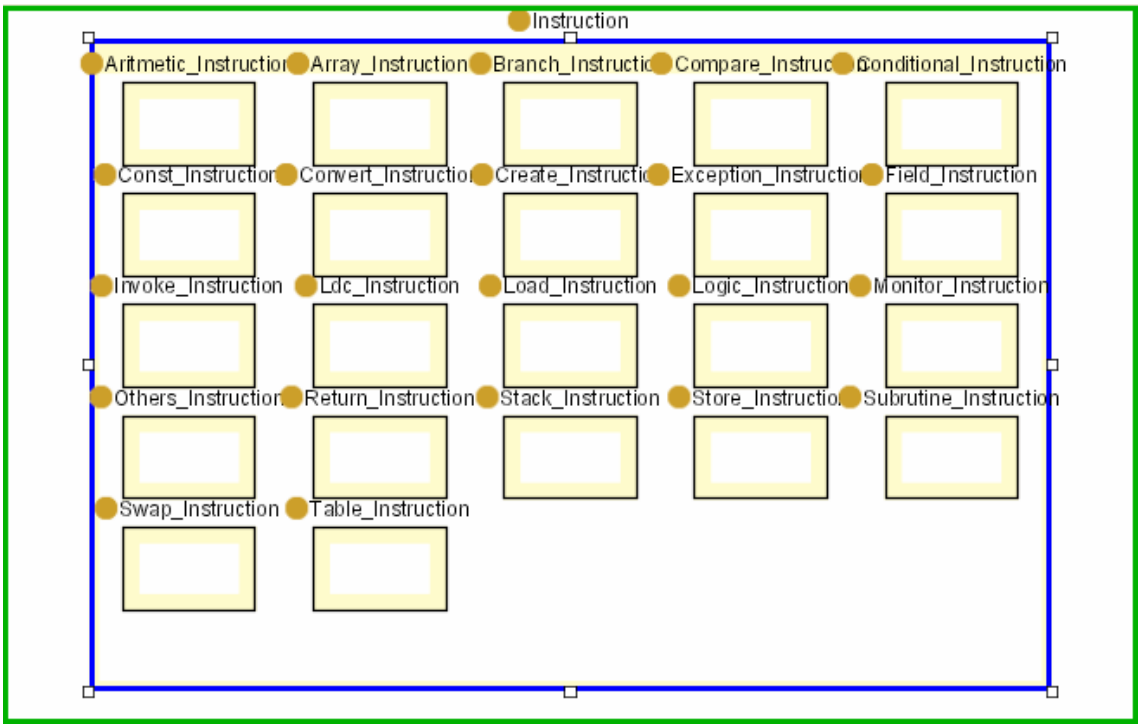
Los **individuos** representan las características concretas de un concepto. Todos los individuos de un concepto tienen las mismas propiedades (slots), pero el valor de dichas propiedades no es el mismo.

Cada instrucción posee un individuo que tiene rellenos en sus slots la información de las dos relaciones anteriores.

Los **conceptos** definen unas características comunes todos los individuos de ese concepto. Se explican gráficamente mediante una extensión de Protégé, denominada Jambalaya.

Primeramente se muestran los temas a los que hace referencia una instrucción. Se pueden distinguir hasta 22 temas diferentes, cada instrucción de la JVM será un descendiente de uno o varios temas. Esta representación tiene

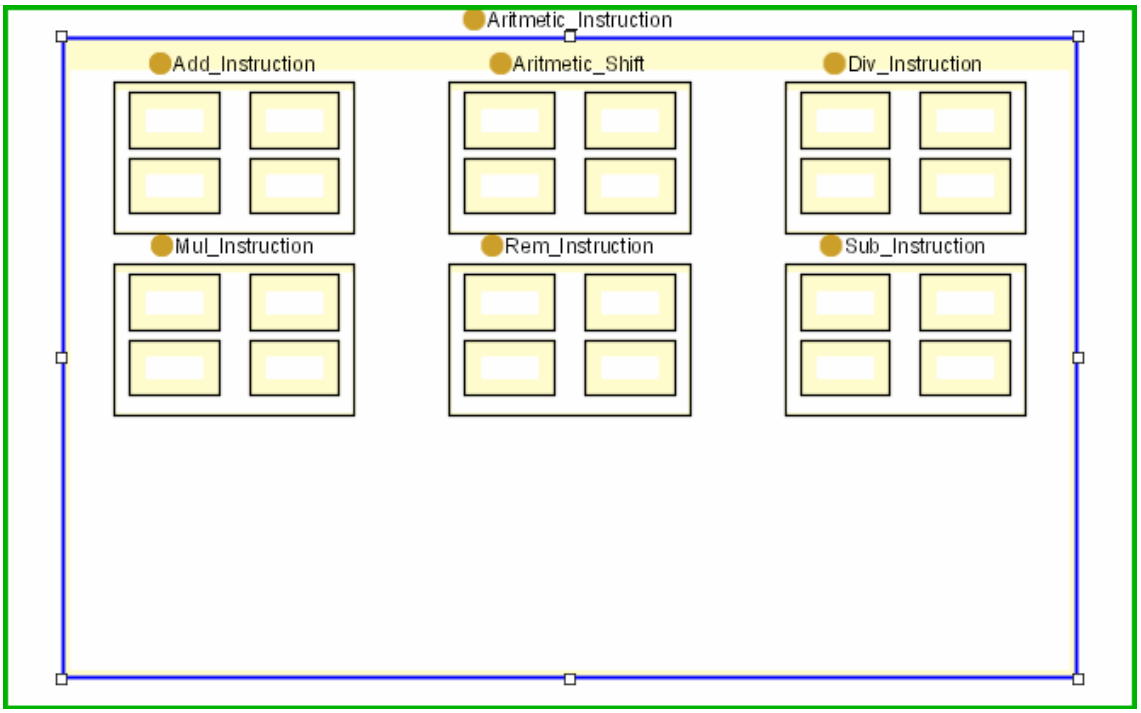
forma de conjuntos, donde el cuadrado azul es el conjunto instrucción, y los cuadrados internos son subconjuntos de instrucción. Los temas principales de las instrucciones son por tanto los hijos directos del concepto ‘Instruction’.



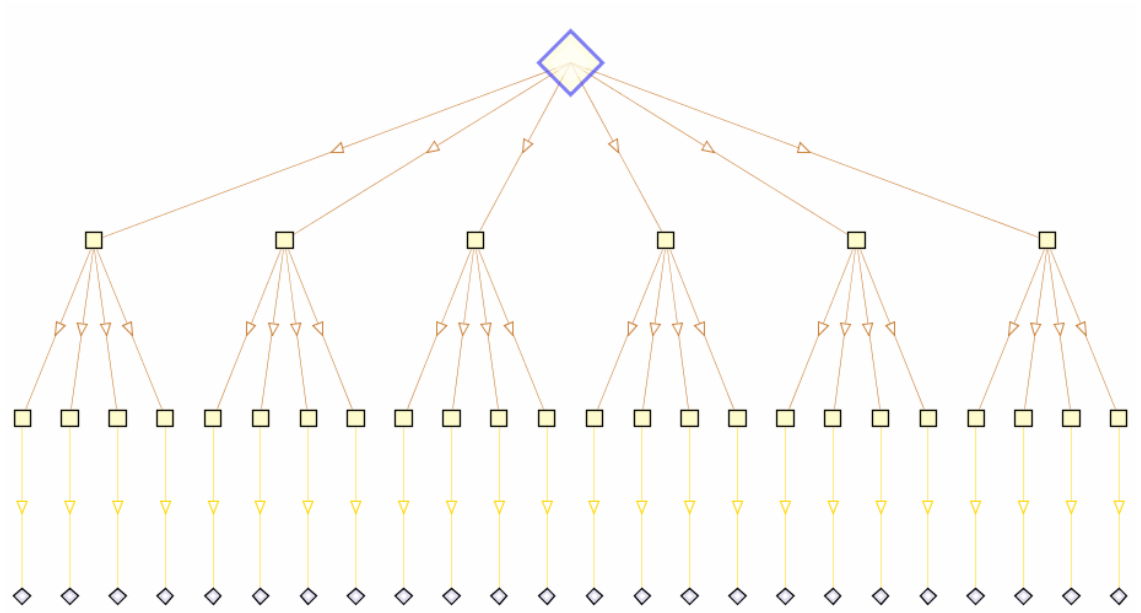
Las siguientes gráficas detallarán uno a uno los temas a los que puede pertenecer una instrucción de la JVM.

Aritmetic_Instruction:

Representan el conjunto de instrucciones aritméticas de la JVM.



Otra representación que permite ver la estructuración que se ha hecho de las instrucciones aritméticas es en forma de árbol. Tanto la figura anterior como la siguiente representan lo mismo, pero son dos formas distintas de verlo.



El nodo raíz representa el concepto ‘aritmetic_instruction’. Los descendientes representan subconceptos de las instrucciones aritméticas, a excepción de las hojas que son individuos, los cuales tienen rellenos sus slots o propiedades con el valor correspondiente. Puesto que Jambalaya no permite mostrar el nombre de todos los conceptos a la vez, a continuación hacemos una clasificación de los mismos, con una breve descripción de las instrucciones.

- Suma.
  - dadd: Suma de elementos de tipo ‘double’.
  - iadd: Suma de elementos de tipo ‘int’.
  - fadd: Suma de elementos de tipo ‘float’.
  - ladd: Suma de elementos de tipo ‘long’.

- Desplazamiento aritmético.
  - ishl: Desplazamiento de un entero a la izquierda.
  - ishr: Desplazamiento de un entero a la derecha.
  - lshl: Desplazamiento de un long a la izquierda.
  - lshr: Desplazamiento de un long a la derecha.

- División.
  - ddiv: División de elementos de tipo ‘double’.
  - idiv: División de elementos de tipo ‘int’.
  - fdiv: División de elementos de tipo ‘float’.
  - ldiv División de elementos de tipo ‘long’.

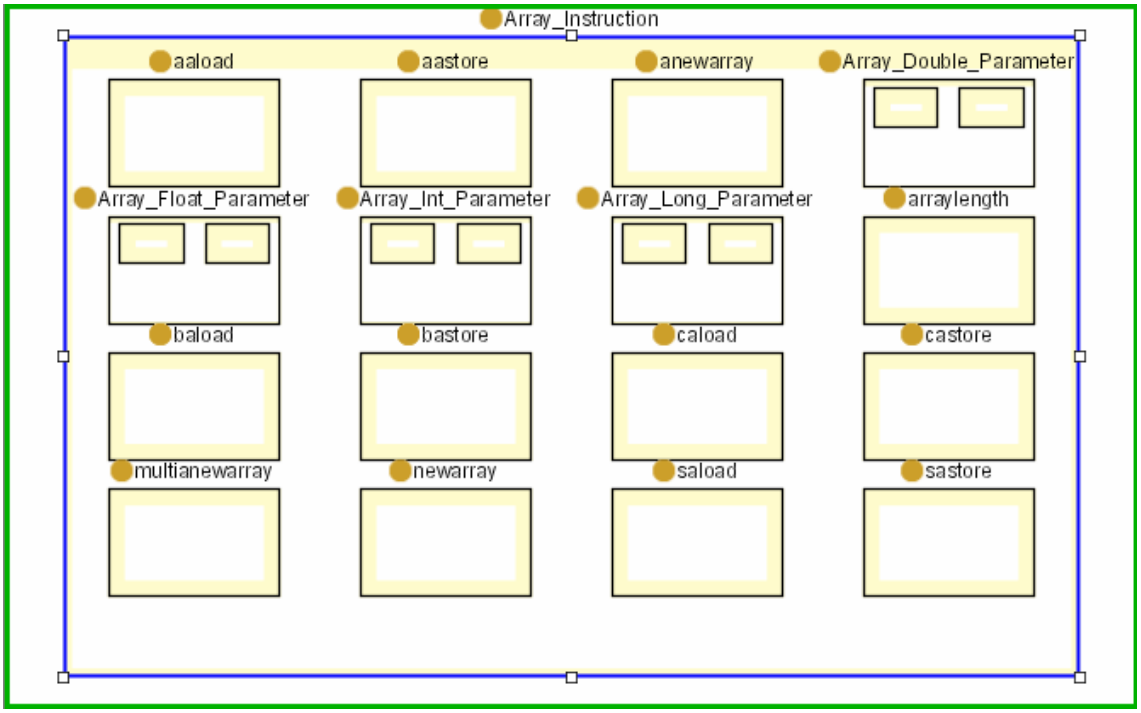
- Multiplicación.
  - dmul: Multiplicación de elementos de tipo ‘double’.
  - imul: Multiplicación de elementos de tipo ‘int’.
  - fmul: Multiplicación de elementos de tipo ‘float’.
  - lmul: Multiplicación de elementos de tipo ‘long’.

- Resto de la división.
    - drem: Resto de la división de elementos de tipo ‘double’.
    - irem: Resto de la división de elementos de tipo ‘int’.
    - frem: Resto de la división de elementos de tipo ‘float’.
    - lrem: Resto de la división de elementos de tipo ‘long’.

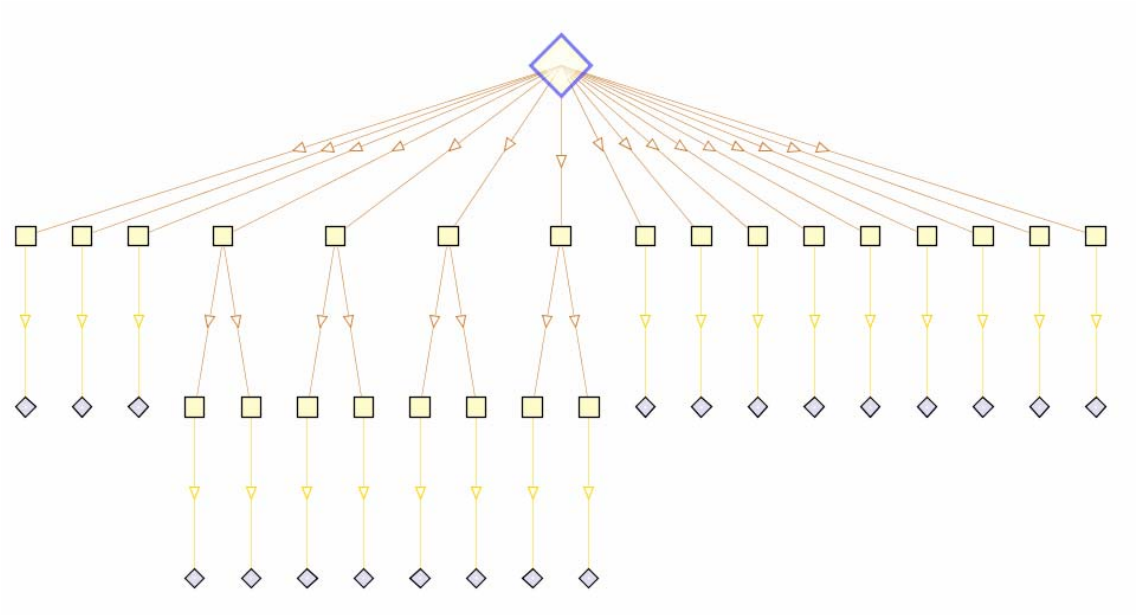
- Resta.
    - dsub: Resta de elementos de tipo ‘double’.
    - isub: Resta de elementos de tipo ‘int’.
    - fsub: Resta de elementos de tipo ‘float’.
    - lsub: Resta de elementos de tipo ‘long’.

Array_Instruction:

Representan al conjunto de instrucciones que permiten lectura y escritura sobre alguna posición de dato de tipo array.



Al igual que para las instrucciones aritméticas, a continuación mostramos en forma de árbol la clasificación de las instrucciones que trabajan con el tipo de datos array.



Como se puede observar en la jerarquía de las instrucciones de array, la mayor parte de los hijos de 'array_instruction' son directamente instrucciones de la JVM y no un subtema, a excepción de 4 categorías, clasificadas por el tipo de parámetro con el que operan. A continuación se detalla la organización de las instrucciones que trabajan con arrays.

- aaload. Obtienen un elemento de un array.
- aastore. Almacena un elemento en un array.
- anewarray. Crea una nueva referencia a un array
- Array_Double_Parameter
    - daload. Obtiene un elemento de un array cuyos elementos son de tipo 'double'.
    - dastore. Almacena un elemento en un array cuyos elementos son de tipo 'double'.

- Array_Float_Parameter
    - faload. Obtiene un elemento de un array cuyo elementos son de tipo ‘float’.
    - fastore. Almacena un elemento en un array cuyos elementos son de tipo ‘float’.

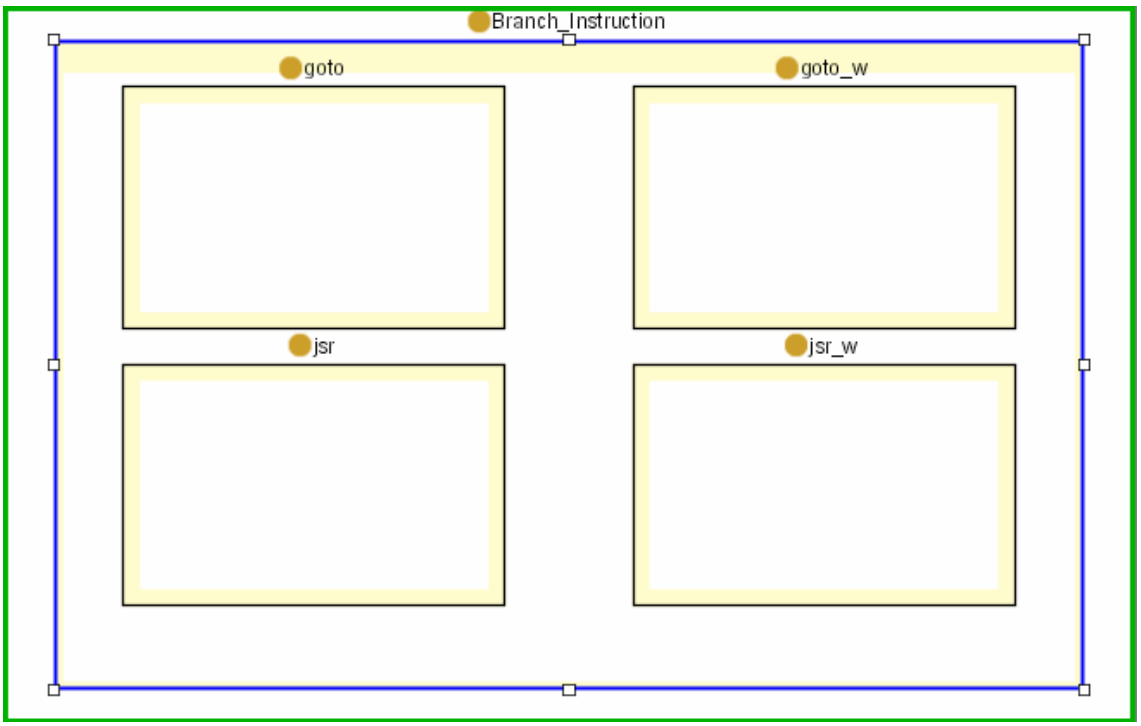
- Array_Int_Parameter
    - iaload. Obtiene un elemento de un array cuyos elementos son de tipo ‘int’.
    - iastore. Almacena un elemento en un array cuyos elementos son de tipo ‘int’.

- Array_Long_Parameter
    - laload. Obtiene un elemento de un array cuyos elementos son de tipo ‘long.
    - lastore. Almacena un elemento en un array cuyos elementos son de tipo ‘long’.

- arraylength. Obtiene la longitud de un array.
- balead Almacena un elemento en un array cuyos elementos son de tipo ‘byte’ ‘boolean’.
- bastore. Obtiene un elemento de un array cuyos elementos son de tipo ‘byte’ o ‘boolean’.
- caload Almacena un elemento en un array cuyos elementos son de tipo ‘char’.
- castore Obtiene un elemento de un array cuyos elementos son de tipo ‘char’.
- multinewarray: Crea un array de varias dimensiones.
- newarray Crea un array de elementos de un tipo determinado
- saload Obtiene un elemento de un array cuyos elementos son de tipo ‘short’.
- sastore Almacena un elemento en un array cuyos elementos son de tipo ‘short’.

Instrucciones de salto:

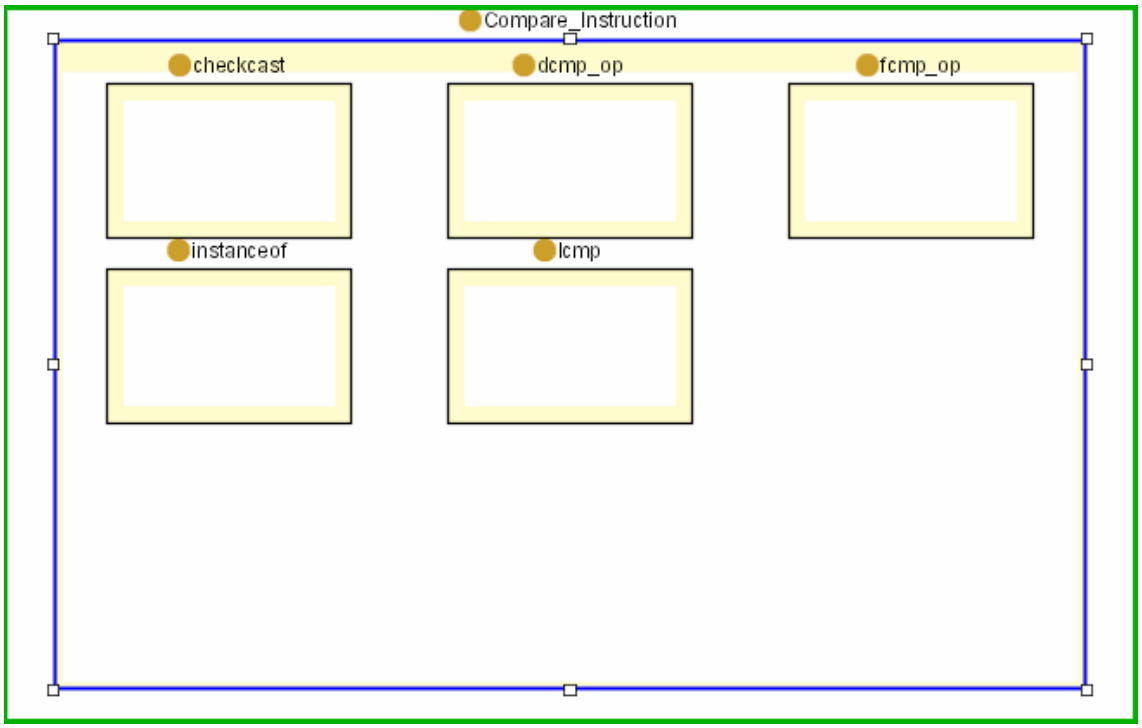
Normalmente la ejecución de un código es secuencial, pero estas instrucciones permiten dar saltos en el orden de ejecución del código, permitiendo saltos hacia adelante o hacia atrás en la secuencia de instrucciones.



- goto: Para saltar a una dirección con desplazamiento de 16bits.
- goto_w: Para saltar a una dirección con desplazamiento de 32bits.
- jsr: Igual que goto pero trabaja con la pila de la JVM.
- jsr_w: Igual que goto_w pero trabaja con la pila de la JVM.

Compare_Instruction:

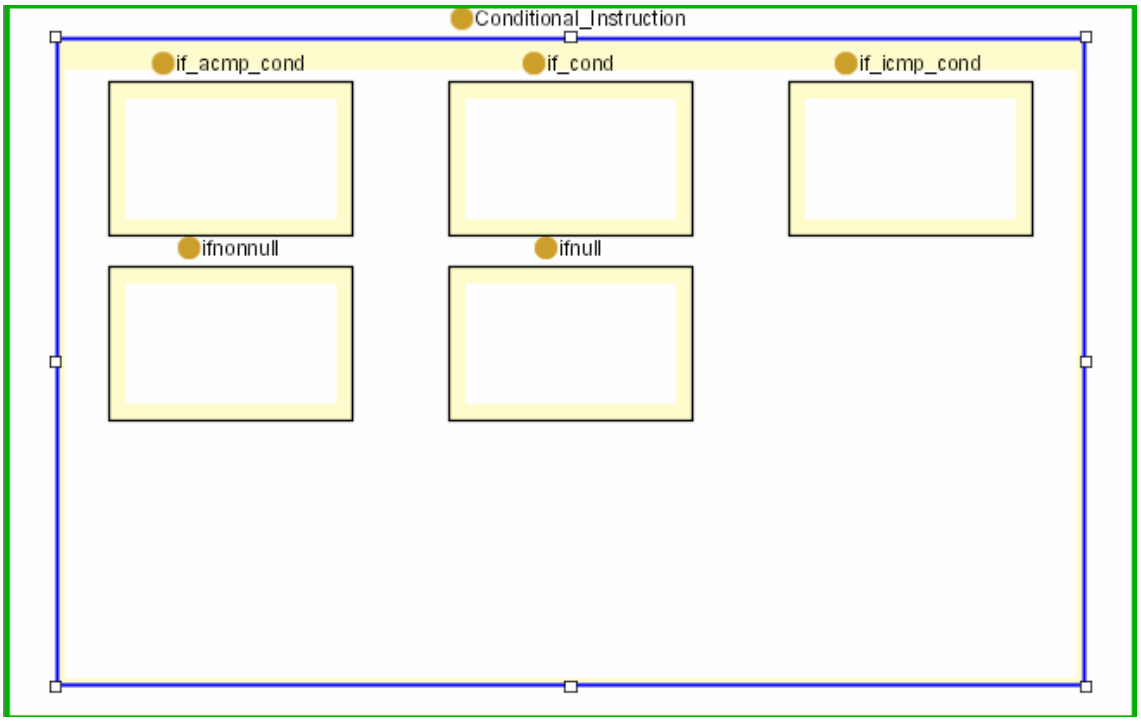
Representa el conjunto de instrucciones que realiza comparaciones, dichas comparaciones son de distintos tipos según la instrucción empleada.



- checkcast: Comprueba si un objeto es de un tipo concreto.
- dcmp_op: Compara dos elementos de tipo double (que están en la cima de la pila de la JVM) según la operación 'op'. 'op' pertenece a {'g' = value1 greater than value2, 'e' = equal, 'l' = value1 less than value2}
- fcmp_op: Igual que 'dcmp_op' pero para elementos de tipo 'float'.
- instanceof: Determina si un objeto es de un tipo de formato concreto.
- lcmp: Igual que 'cmp_op' pero para elementos de tipo 'long'.

Condicional_Instruction:

Las instrucciones condicionales son aquellas que ejecutan un código u otro en función a si se cumple la condición o no.



- if_acmp_cond: Salta si se cumple la condición entre el formato de dos elementos.
- if_cond: Salta si se cumple la condición.
  Cond:
  - eq: = 0
  - ne: =|= 0
  - lt: < 0
  - le: <= 0
  - gt: > 0
  - ge : >= 0
- if_icmp_cond: Salta si se cumple la condición entre dos elementos de tipo ‘int’.
  Cond:
  - eq: = 0

ne: =|= 0

lt: < 0

le: <= 0

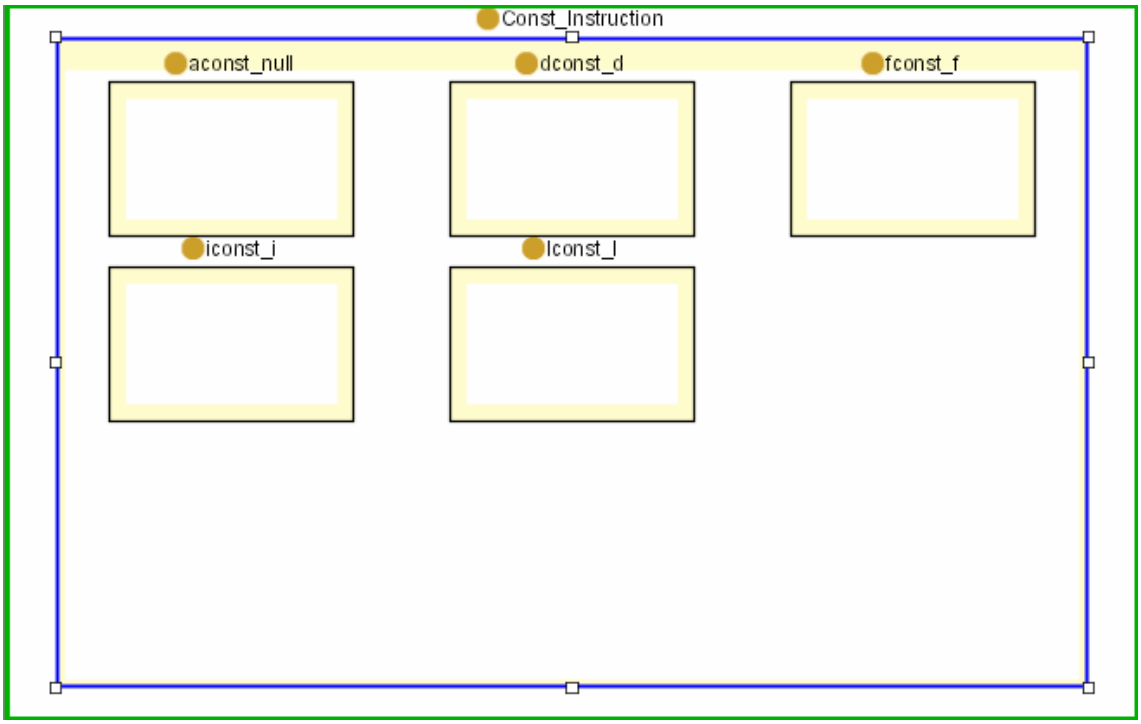
gt: > 0

ge : >= 0

- ifnonnull: Salta si la referencia no es null.
- ifnull: Salta si la referencia es null.

Const_Instruction:

Estas instrucciones se usan para apilar constantes en la pila.



El formato es el siguiente:

xconst_y donde x e y pertenece a {d, f, i, l}

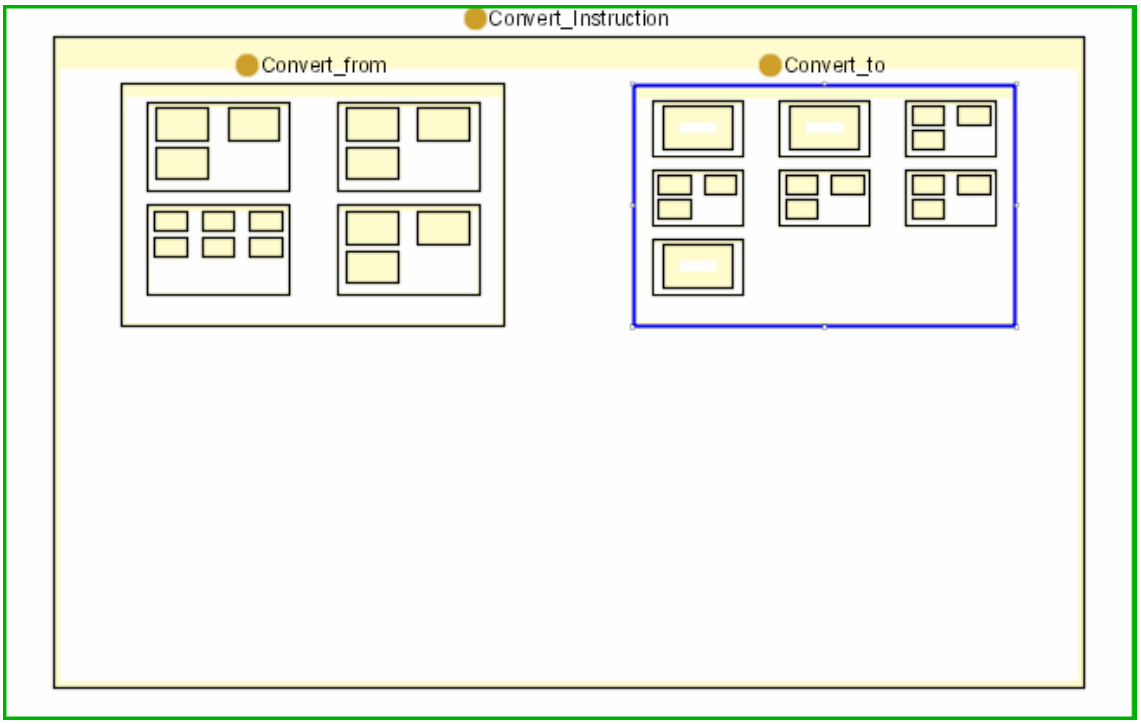
d = double
f = float
i = int
l = long

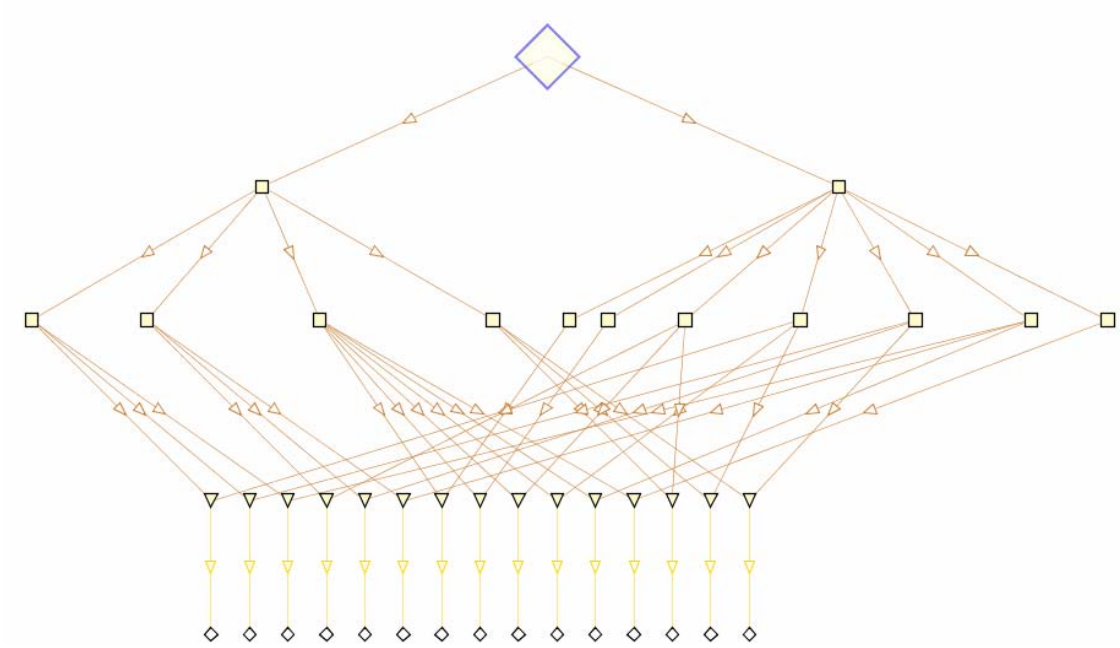
La ‘y’ es un número del tipo al que pertenece, por ejemplo iconst_3. Las instrucciones para manejo de constantes son las siguientes:

- aconst_null:
- dconst_d:
- fconst_f:
- iconst_i:
- lconst_l:

Instrucciones de conversión de tipos:

Permiten convertir de un tipo a otro siempre que sea compatible. Los tipos a convertir tienen que ser numéricos.





En esta representación podemos observar que hay hijos con varios padres, esto es porque una instrucción que convierte, por ejemplo, un número de tipo ‘float’ a tipo ‘int’ tendrá como padre convierte a ‘int’ y convierte un ‘float’.

La sintaxis de estas instrucciones es:

x2y, donde ‘x’ e ‘y’ pertenecen a {d, f, i, l}

d = double
f = float
i = int
l = long

Así pues f2i sería una instrucción que convierte de ‘float’ a ‘int.’
A continuación se detallan los nodos del árbol.

- Convert_from.
  - Convert_from_double. Convierten de ‘double’ a otro tipo.
    - d2f.
    - d2i.
    - d2l.
  - Convert_from_float. Convierten de ‘float’ a otro tipo.
    - f2d
    - f2i.

        - f2l.

    - Convert_from_int. Convierten de ‘int’ a otro tipo.
        - i2d
        - i2f
        - i2l

    - Convert_from_long. Convierten de ‘long’ a otro tipo.
        - l2d
        - l2f
        - l2i

- Convert_to.
    - Convert_to_double. Convierten de un tipo a ‘double’.
        - f2d
        - i2d
        - l2d

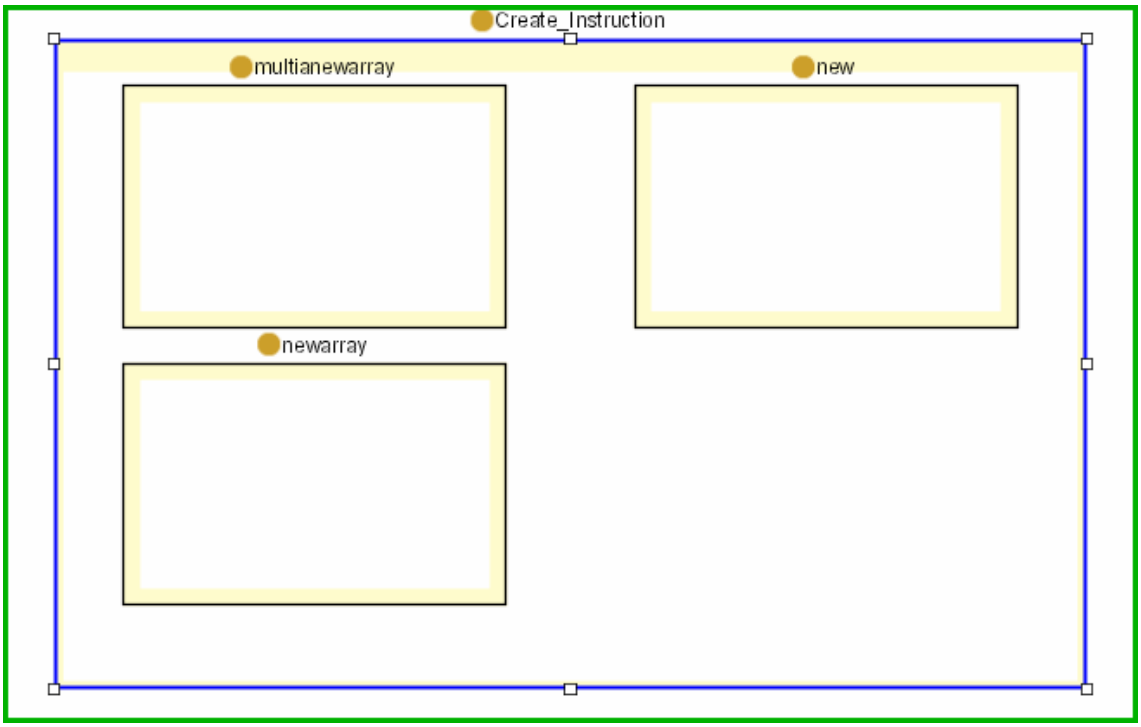
    - Convert_ to_float. Convierten de un tipo a ‘float’.
        - d2f
        - i2f
        - l2f

    - Convert_ to_int. Convierten de un tipo a ‘int’.
        - d2i
        - f2i
        - l2i

    - Convert_ to_long. Convierten de un tipo a ‘long’.
        - d2l
        - f2l
        - i2l

Create_Instruction:

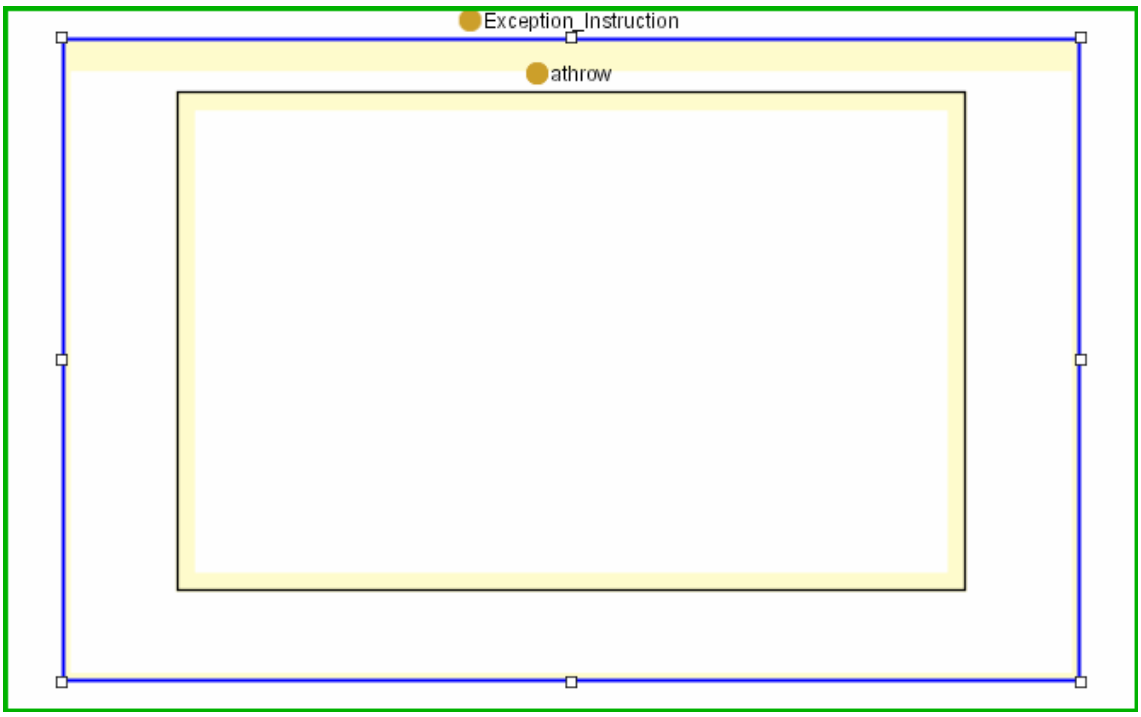
Representa el conjunto de instrucciones que crean un elemento, es decir, reserva el espacio de memoria que ocupe el objeto creado, y crea una referencia a esa posición.



- multinewarray. Crea un array multidimensional.
- new. Crea un nuevo objeto.
- newarray. Crea un objeto de tipo array.

Exception_Instruction:

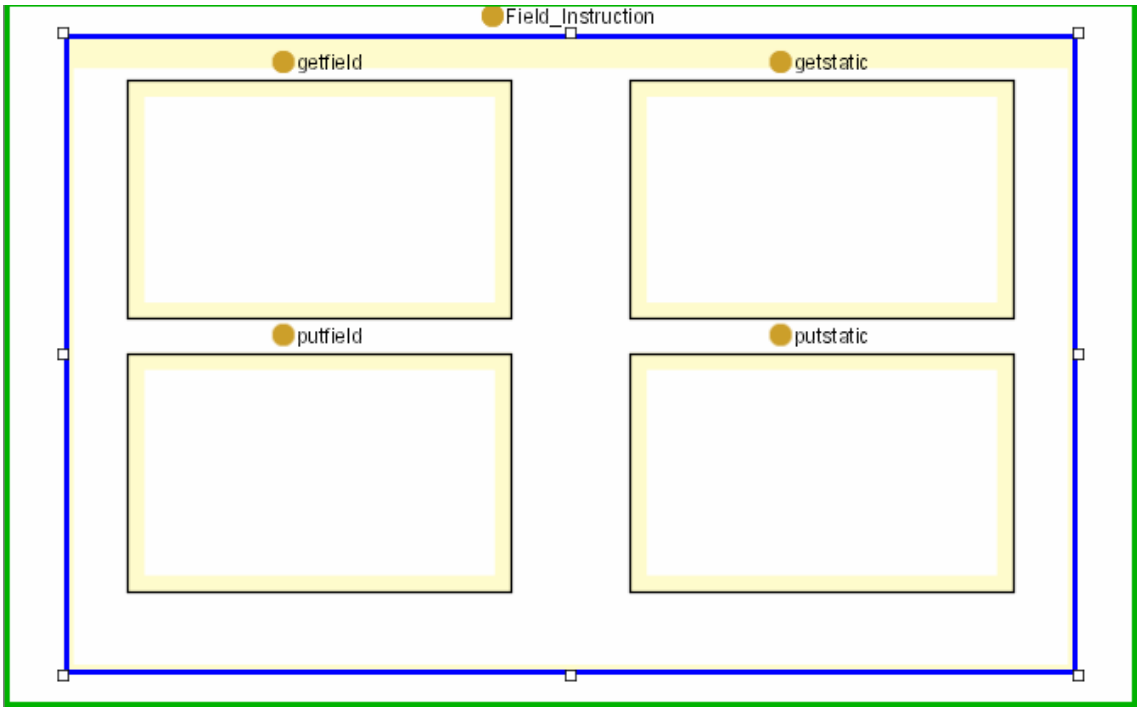
Instrucciones que lanzan error o excepción.



- athrow: Lanza una excepción o un error.

Field_Instruction:

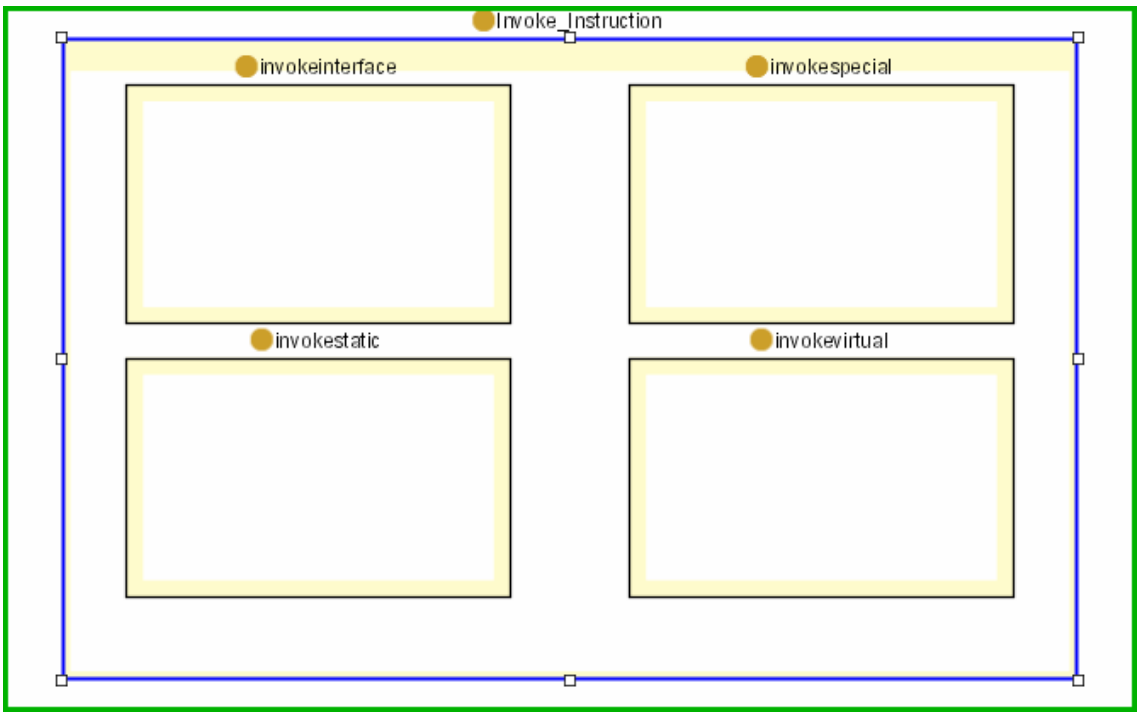
Representa el conjunto de instrucciones que permiten acceso y modificación de los campos de un objeto o una clase.



- getfield: Busca el campo de un objeto,
- getstatic: Busca el campo estático de una clase.
- putfield: Pone un campo en un objeto
- putstatic: Pone el campo estático de una clase.

Invoke_Instruction:

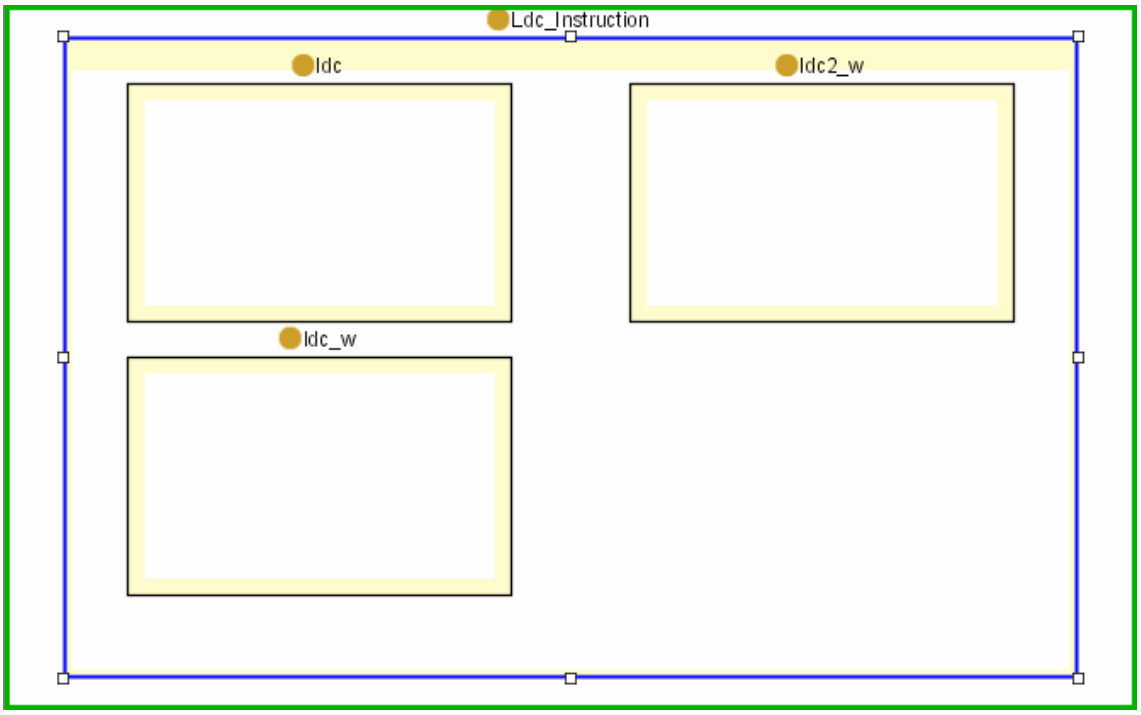
Permite invocar métodos de distintas formas.



- invokeinterface: Invoca un método implementado de un interfaz.
- Invokespecial: Invoca un método que necesita tratamiento especial
- Invokestatic: Invoca un método estático.
- Invokevirtual: Invoca un método de una instancia de una clase, es la forma habitual de invocación.

Ldc_Instuction:

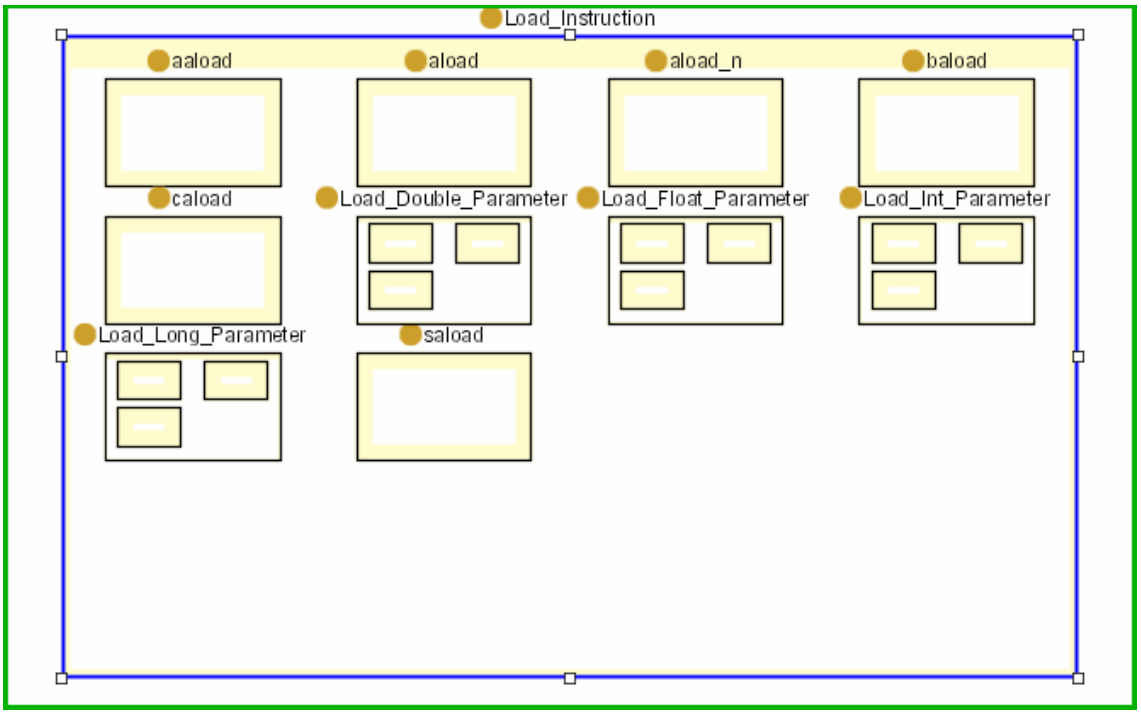
Permite la carga de constantes a la pila con direccionamiento indirecto, usando la ‘Constant Pool’.

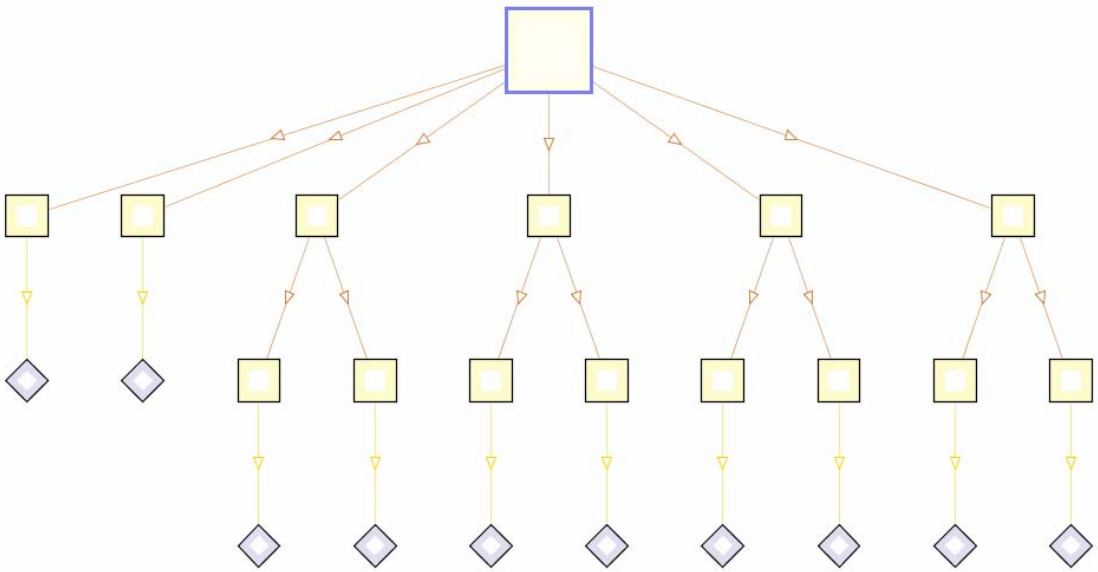


- ldc: Carga una constante de 32 bits a la pila
- ldc2_w: Carga una constante de 64 bits a la pila.
- ldc_w: Carga una constante a la pila, con indexación distinta que en ldc.

Load instruction:

Representa el conjunto de instrucciones de la JVM que permiten leer o cargar datos de una variable





- aload: Carga una referencia a una variable en la pila.
- aload_n Carga una referencia a una variable en la pila.

- Load_Double_Parameter
    - dload Coloca en la cima de la pila el valor de una variable de tipo ‘double’.
    - dload_n Coloca en la cima de la pila el valor de una variable de tipo ‘double’.

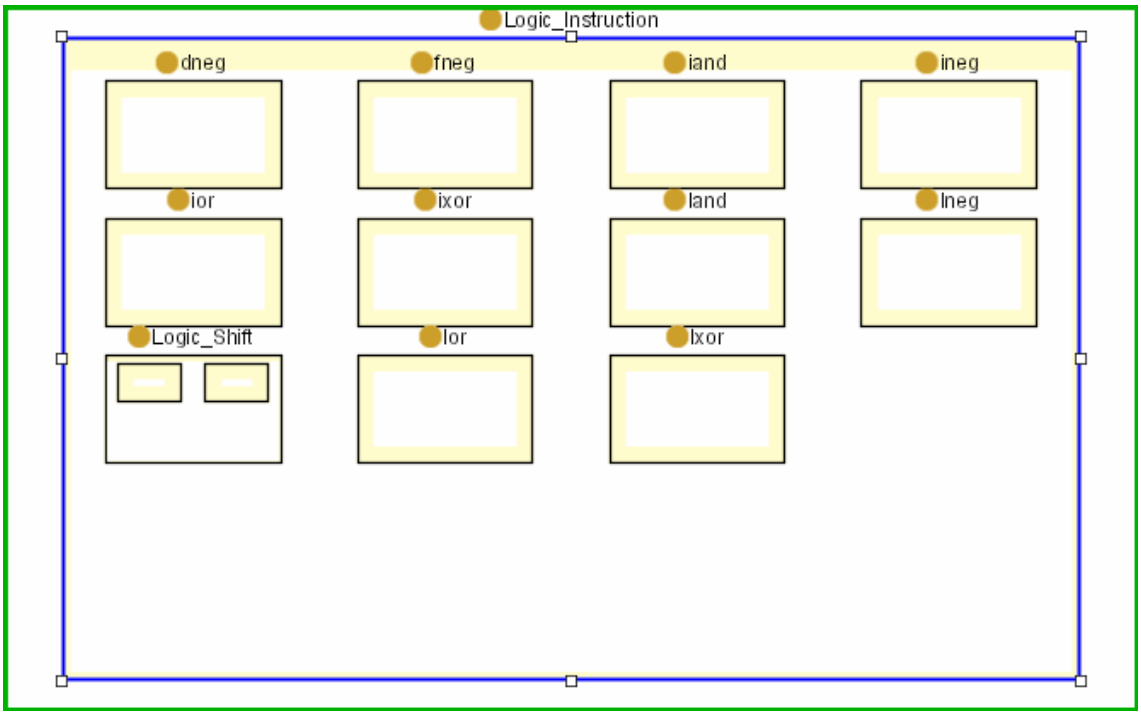
- Load_Float_Parameter
    - fload Coloca en la cima de la pila el valor de una variable de tipo ‘float’.
    - fload_n Coloca en la cima de la pila el valor de una variable de tipo ‘float’.

- Load_Int_Parameter
    - iload Coloca en la cima de la pila el valor de una variable de tipo ‘int’.
    - iload_n Coloca en la cima de la pila el valor de una variable de tipo ‘int’.

- Load_Long_Parameter
    - lload Coloca en la cima de la pila el valor de una variable de tipo ‘long’.
    - lload_n Coloca en la cima de la pila el valor de una variable de tipo ‘long’.

Logic instuction:

Representa el conjunto de instrucciones que realizan operaciones lógicas entre datos de tipo numérico.



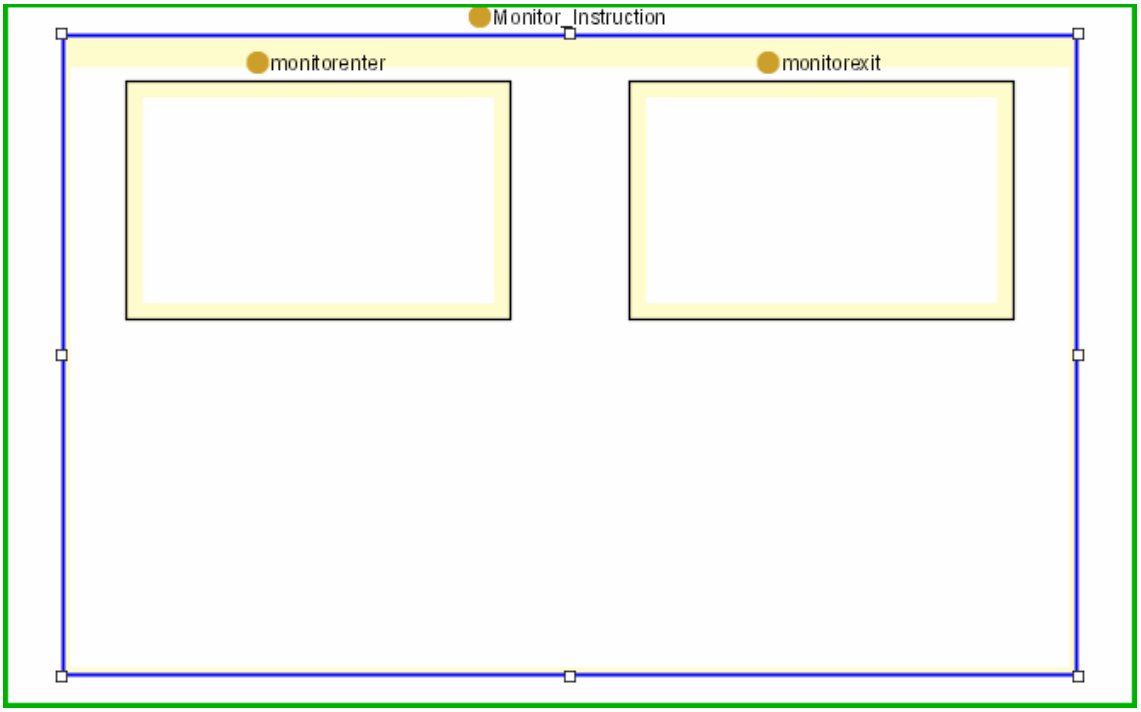
- dneg: Niega un elemento de tipo ‘double’.
- fneg: Niega un elemento de tipo ‘float’.
- iand: Implementa la ‘and’ entre dos elementos de tipo ‘int’.
- ineg: Niega un elemento de tipo ‘int’.
- ior: Implementa la ‘or’ entre dos elementos de tipo ‘int’.
- ixor: Implementa la ‘xor’ entre dos elementos de tipo ‘int’.
- land: Implementa la ‘and’ entre dos elementos de tipo ‘long’.
- lneg: Niega un elemento de tipo ‘long’.
- Logic_Shift
    - iushr: Desplazamiento lógico a la derecha de un entero.
    - lushr: Desplazamiento lógico a la derecha de un elemento de tipo ‘long’.

- lor: Implementa la ‘or’ entre dos elementos de tipo ‘long’.

- lxor: Implementa la ‘xor’ entre dos elementos de tipo ‘long’.

Monitor instruction:

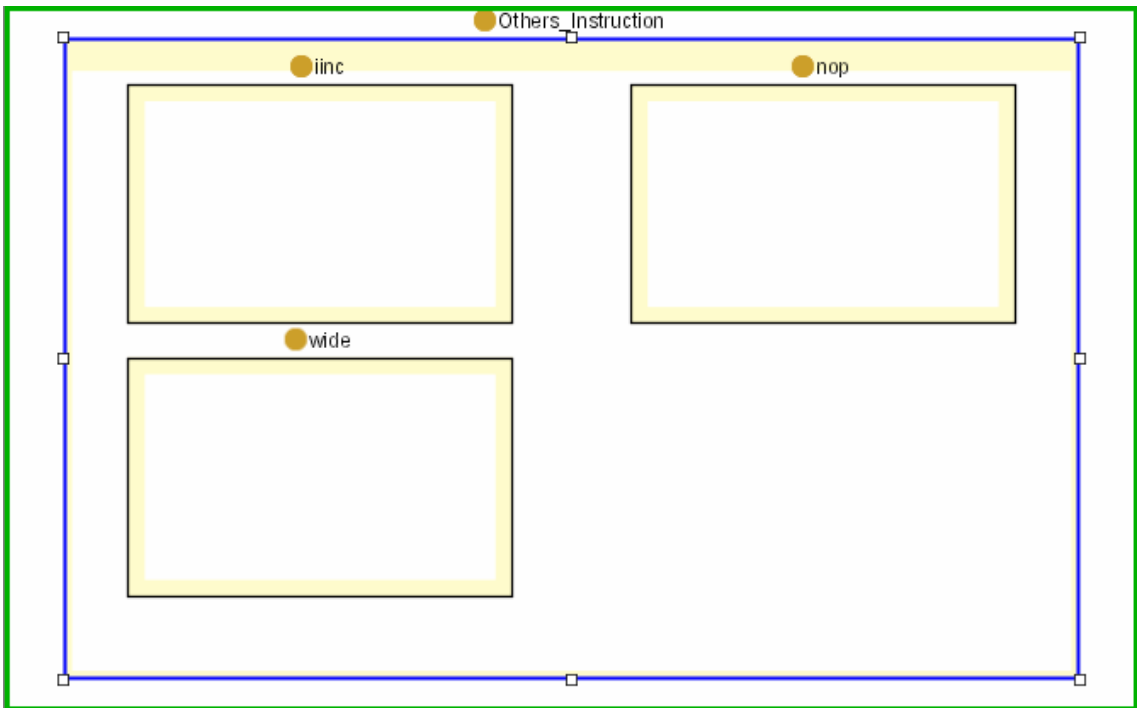
Representa las instrucciones que permiten trabajar con monitores Java.



- monitorenter: Entra en el monitor asociado a un objeto.
- monitorexit: Sale del monitor asociado a un objeto.

Others instruction:

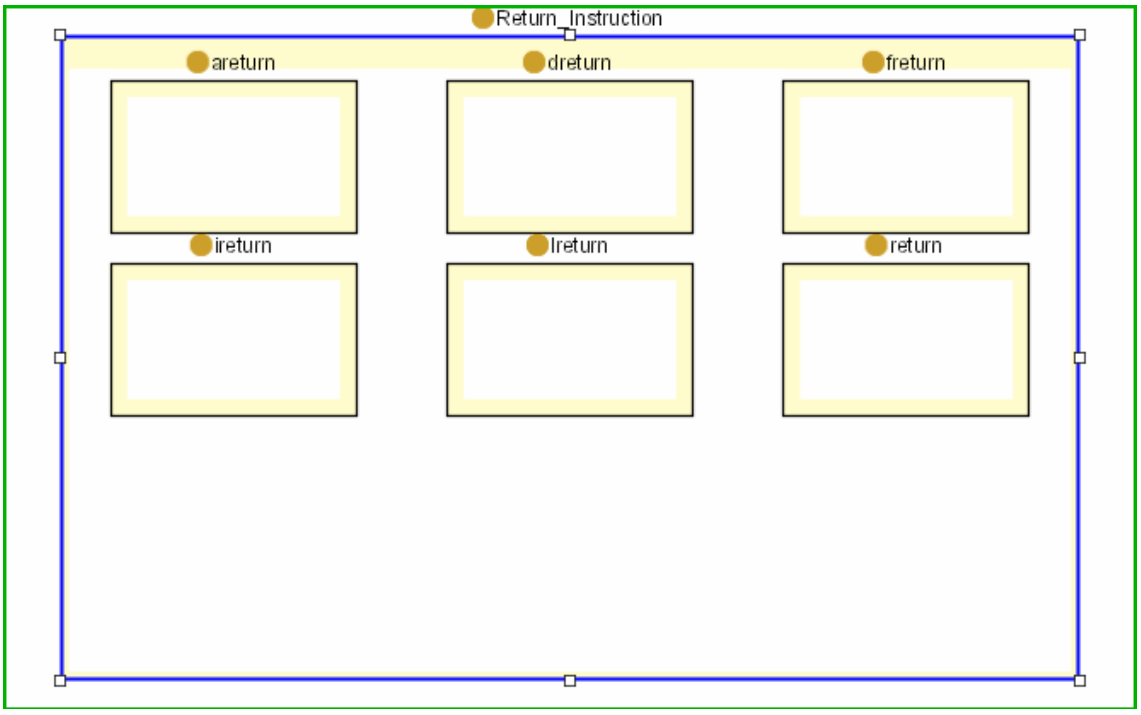
Representa el conjunto de instrucciones que no se pueden clasificar con los temas principales.



- iinc Incrementa una variable local en una constante.
- nop: No operación, no hace nada.
- wide: Permite modificar el comportamiento de otras instrucciones: iload, fload, aload, lload, dload, istore, fstore, astore, lstore, dstore, o ret.

Return instruction:

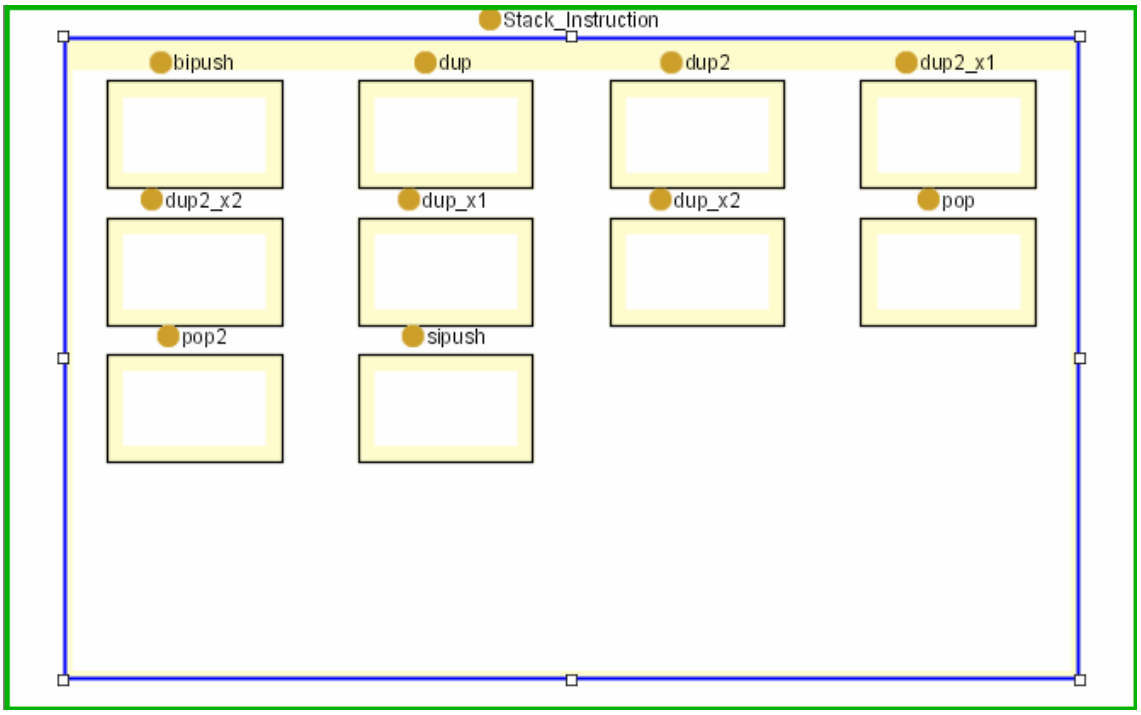
Representa el conjunto de instrucciones que informan de que un método ha terminado, y devuelven el valor del resultado de la ejecución de dicho método.



- areturn: Para devolver el resultado de un método. Este resultado es una referencia a un objeto.
- dreturn: Para devolver el resultado de un método. Este resultado es de tipo ‘double’.
- freturn: Para devolver el resultado de un método. Este resultado es de tipo ‘float’.
- ireturn: Para devolver el resultado de un método. Este resultado es de tipo ‘int’.
- lreturn: Para devolver el resultado de un método. Este resultado es de tipo ‘long’.
- return: Para devolver el resultado vacío de un método. Es decir el método no es una función sino un procedimiento.

Stack instruction.

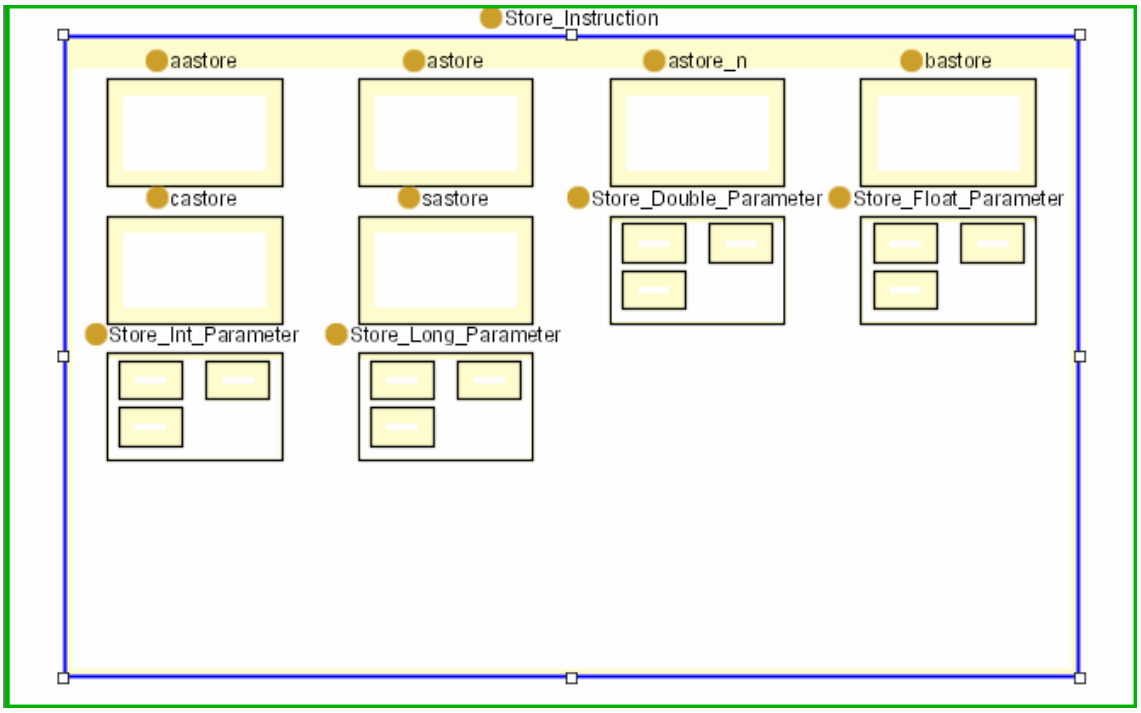
Representa el conjunto de instrucciones que realizan operaciones directas con la pila de la JVM.

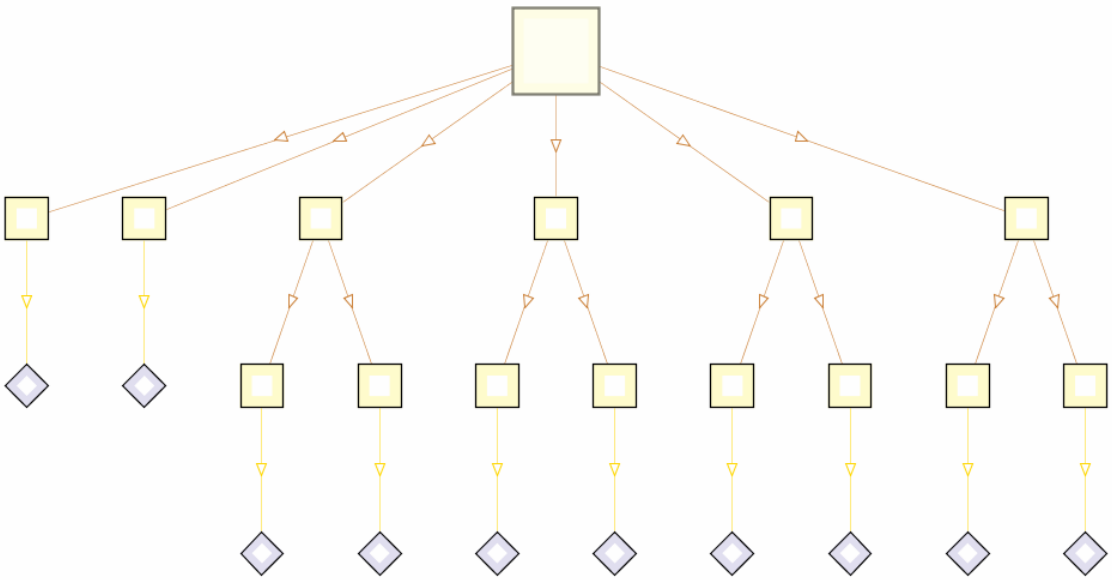


- bipush: Apila un elemento de tipo ‘byte’ en la cima de la pila.
- dup: Duplica el operando de la cima de la pila.
- dup2: Duplica los dos operandos de la cima de la pila.
- dup2_x1: Duplica los dos operandos de la cima de la pila y lo apila dos valores por debajo de la cima.
- dup2_x2: Duplica los dos operandos de la cima de la pila y lo apila dos o tres valores por debajo de la cima.
- dup_x1: Duplica el operando de la cima de la pila y lo apila dos valores por debajo de la cima.
- dup_x2: Duplica el operando de la cima de la pila y lo apila dos o tres valores por debajo de la cima.
- pop: Desapila el valor de la cima de la pila.
- pop2: Desapila lo dos valores de la cima de la pila.
- sipush: Apila un elemento de tipo short.

Store instruction:

Representa el conjunto de instrucciones que permiten la escritura o almacenamiento de un valor en una variable.





- astore: Carga una referencia en una variable.
- astore_n: Carga la referencia en una variable.

- Store_Double_Parameter
  - dstore Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘double’.
  - dstore_n Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘double’.

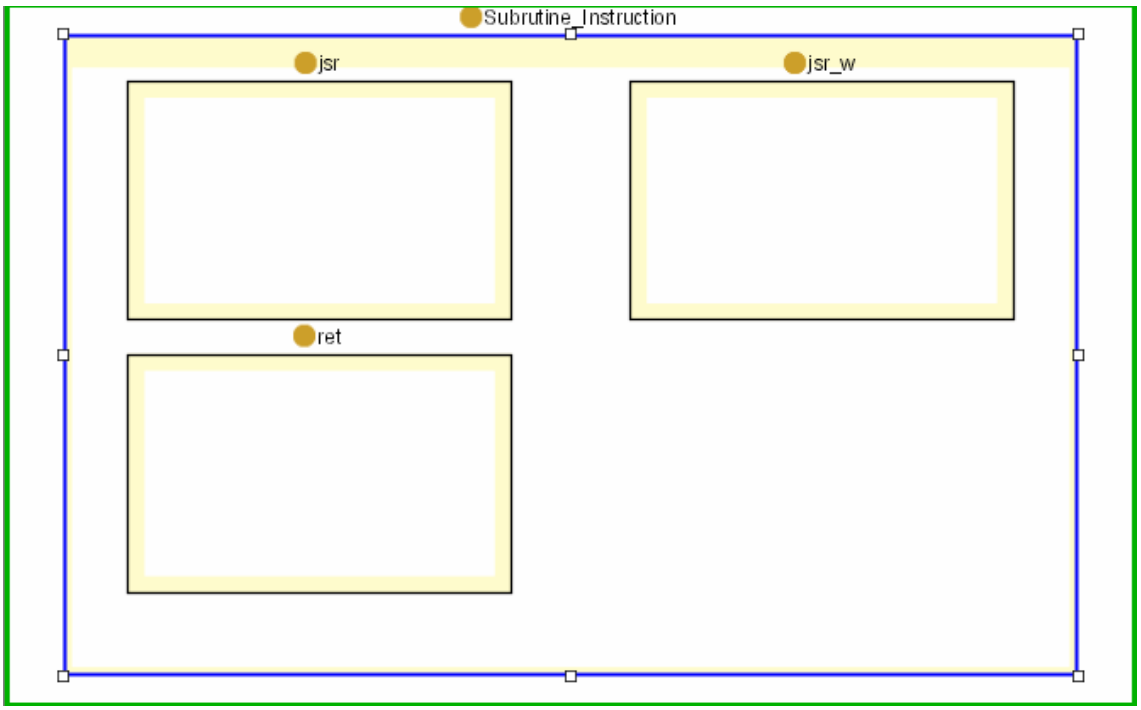
- Store_Float_Parameter
  - fstore Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘float’.
  - fstore_n Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘float’.

- Store_Int_Parameter
  - istore Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘int’.
  - istore_n Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘int’.

- Store_Long_Parameter
  - lstore Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘long’.
  - lstore_n Toma el valor de la cima de la pila y lo asocia a una variable de tipo ‘long’.

Subrutime instruction:

Java trabaja con subrutinas, es decir con programas especiales, las instrucciones que aquí se definen permiten el salto a subrutinas o el retorno de las mismas.



- jsr y jsr_w son instrucciones de salto descritas anteriormente en ‘branch_instruccions’.
- ret Retorno de subrutina.

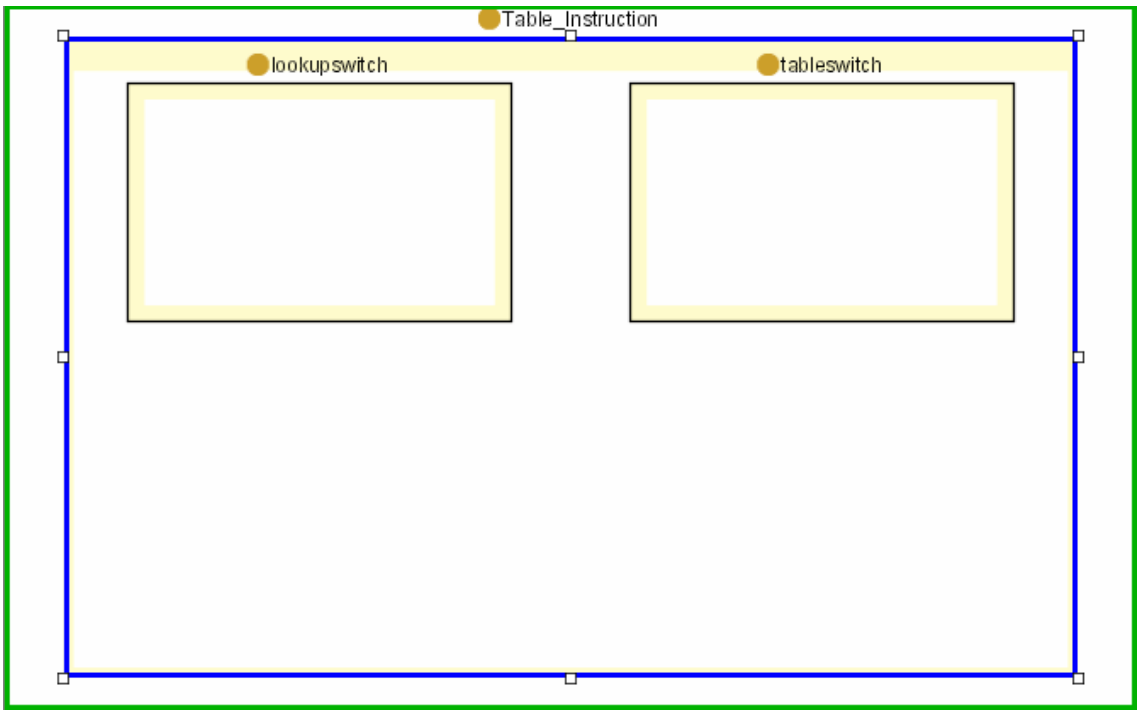
Swap instruction:

Permite intercambiar los valores de la cima de la pila.

- swap: Intercambia los dos valores de la cima de la pila.

Table instruction:

Permite realizar saltos condicionales complejos



- lookupswitch: Implementa la estructura switch/case de java.
- tableswitch: Otra estructura condicional compleja, con más de dos posibilidades de salto.

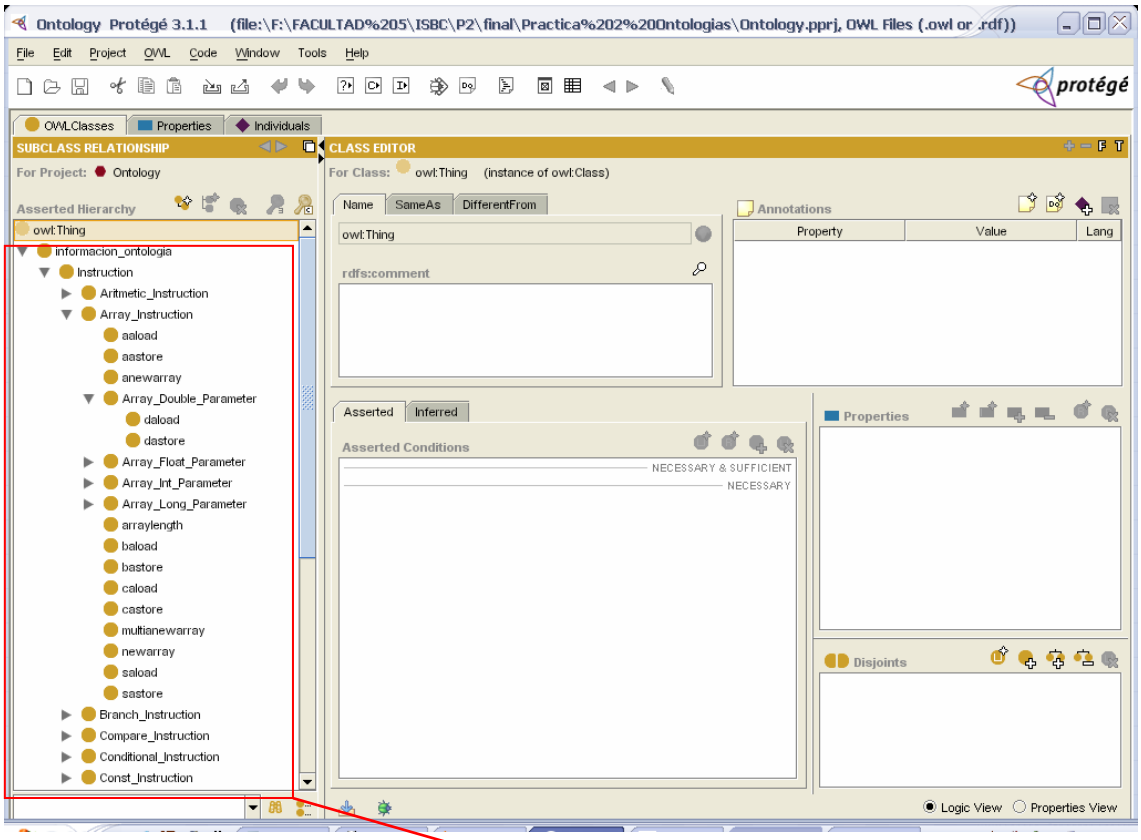
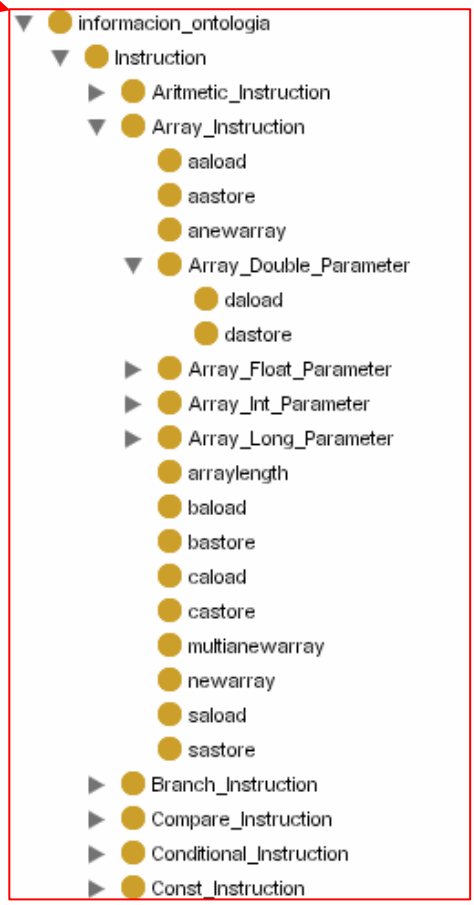


Figura 4.1: Detalle de la ontología



CAPÍTULO 5
FASE DE DISEÑO

5. FASE DE DISEÑO

5.1 INTRODUCCIÓN

Basándonos en los resultados obtenidos en la fase de análisis y teniendo en cuenta las herramientas que se van a usar, se pretende obtener un diseño detallado de la aplicación.

El diseño de una aplicación está orientado a la implementación, y define varios elementos:

- Clases que intervienen.
- Atributos de cada clase.
- Métodos de cada clase.
- Comunicación entre las clases del sistema

Para identificar las clases que van a intervenir en el sistema, así como sus funcionalidades se ha empleado una técnica basada en tarjetas CRC, que se detallan en los apartados siguientes.

Una vez definidas las clases y sus funciones se ha empleado la herramienta Racional-Rose para definir el diseño completo de la aplicación.

5.2 EDITOR DE EJERCICIOS Y CASOS

5.2.1 Introducción.

Para realizar el diseño del módulo partimos de los datos obtenidos del análisis y realizamos una tarjeta CRC para cada una de las posibles clases del módulo, posteriormente realizamos un diseño preliminar en UML y comenzamos a desarrollar el módulo, durante el desarrollo se puede seguir cambiando el diseño si surgen nuevas necesidades no previstas. Al ser un módulo con interfaces gráficos hemos intentando separar claramente las

componentes graficas de sus funcionalidades y los datos que representan, siguiendo el patrón modelo vista controlador (MVC). (Capitulo 7.2.1.2)

5.2.2 Tarjetas CRC.

Para poder realizar el diseño es necesario identificar previamente las clases que van a intervenir en el editor de ejercicios y casos. Para ello se llevan a cabo reuniones. Con este propósito se decide usar una técnica de diseño basada en tarjetas CRC. Cada tarjeta identifica a una clase, y en ella se define la funcionalidad que va a tener y el resto de clases con las que se va a comunicar.

Tarjeta CRC

Nombre: MainFuncionamientoProfesor

Funciones:
- **C**rear la ventana del editor y casos
- Realizar las llamadas a clases principales de los otros módulos que necesiten las clases de este paquete.

Relación:
- MainCBR
- VentanaProfesor

Tarjeta CRC

Nombre: VentanaProfesor

Funciones:
- Contiene las funcionalidades del interfaz grafico del profesor (editor de ejercicios y casos)
- Extiende VentanaProfesorGUI

Relación:
- VentanaProfesorGUI
- SelectorAyuda
- SelectorCaso
- AyudasCaso
- CodigoCompilado
- CodigoErroneo
- MainFuncionamientoProfesor

Tarjeta CRC

Nombre: VentanaProfesorGUI

Funciones:
- Contiene las componentes graficas necesarias para la crear y modificar ejercicios y casos.

Relación:
- VentanaProfesor

Tarjeta CRC

Nombre: CodigoCompilado

Funciones:
- Permite mostrar el código compilado en el editor de casos

Relación:
- VentanaProfesor

Tarjeta CRC

Nombre: CodigoErroneo

Funciones:
- Permite gestionar el código erróneo en el editor de casos

Relación:
- VentanaProfesor

Tarjeta CRC

Nombre: SelectorAyuda

Funciones:
- Permite elegir una ayuda

Relación:
- VentanaProfesor

Tarjeta CRC

Nombre: SelectorCaso

Funciones:
- Permite elegir un caso CBR de la base de casos

Relación:
- VentanaProfesor

Tarjeta CRC

Nombre: AyudasCaso

Funciones:
- Permite gestionar las ayudas que incluyen los casos en el editor de casos

Relación:
- VentanaProfesor

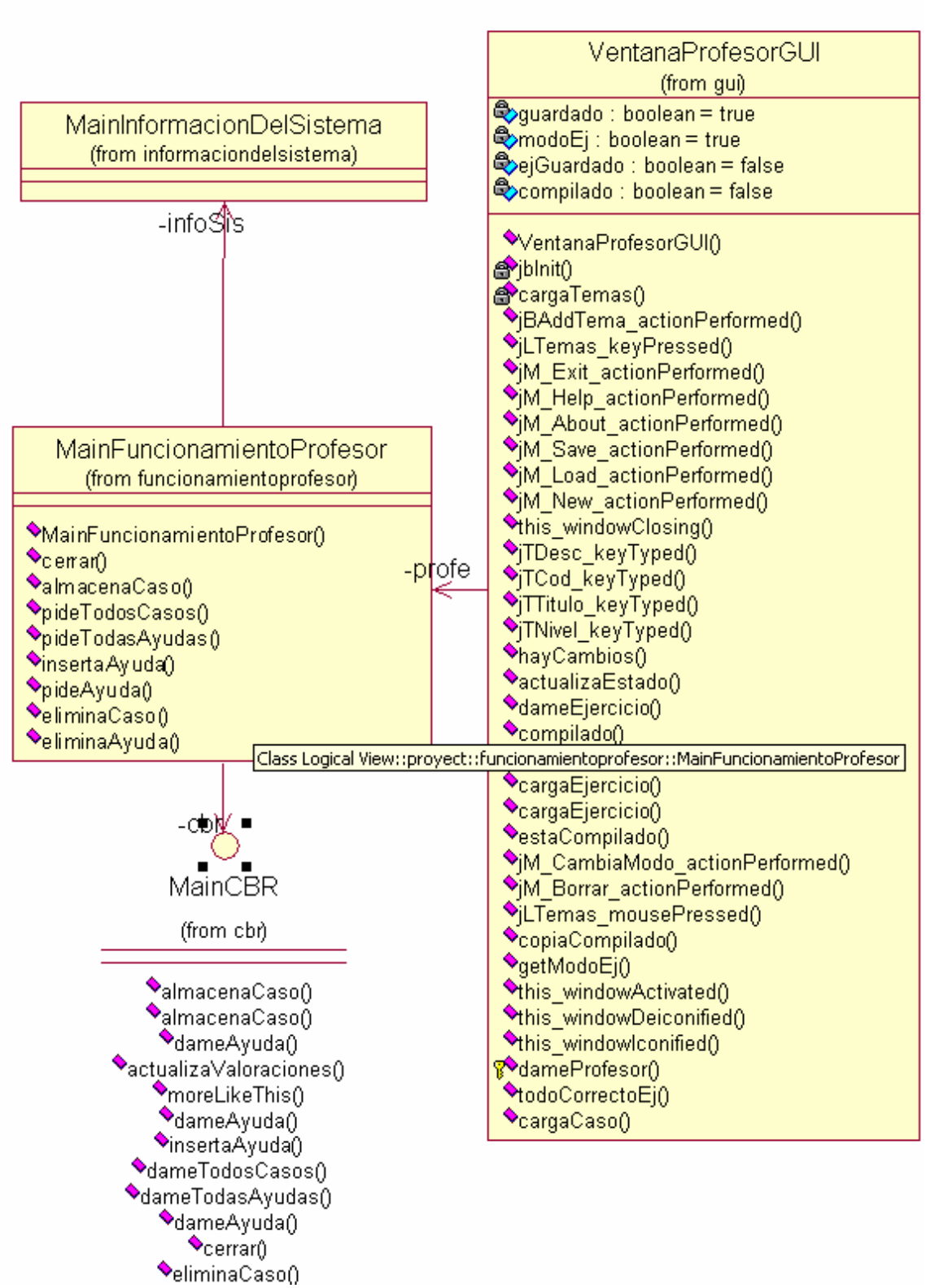
5.2.3 Diseño UML

Una vez identificadas las clases con sus funciones, realizar el diseño es una labor más sencilla. El diseño entra en detalles de implementación, define los métodos de cada clase y sus atributos.

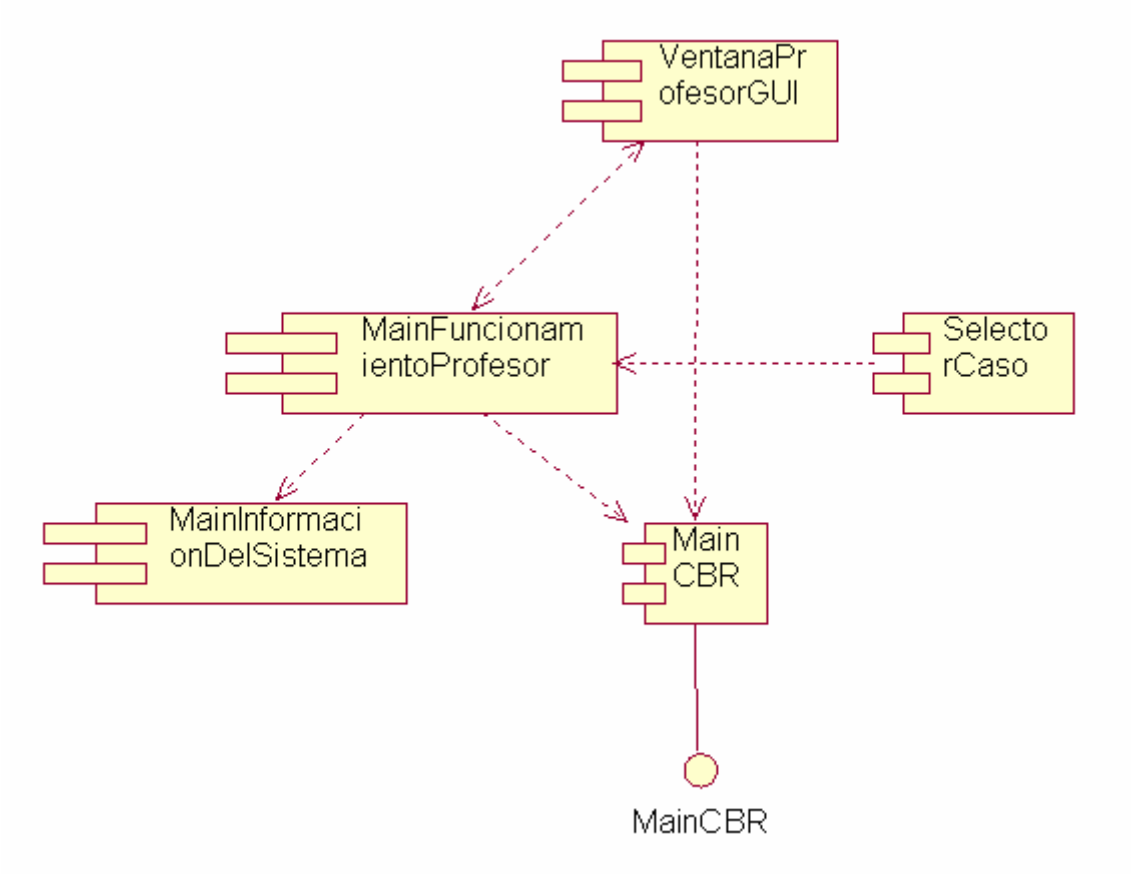
El proceso es iterativo, se parte de una tarjeta CRC, y se añaden los métodos que le permiten cumplir las funciones definidas en la tarjeta, los atributos son las estructuras de datos que darán sopote a estas funcionalidades.

Una vez que cumple las funciones especificadas se incluyen los métodos que permitan la comunicación con las clases con las que debe comunicarse.

A continuación se muestra el diseño de cada una de las clases definidas por las tarjetas CRC.



El gráfico a continuación permite ver la comunicación entre las clases principales del editor de ejercicios y casos.



5.3 MÓDULO DE APRENDIZAJE

5.3.1 Introducción.

Partiendo de los datos obtenidos del análisis y realizamos una tarjeta CRC para cada una de las posibles clases del módulo, posteriormente realizamos el diseño en UML. Este módulo también contiene interfaces gráficos y también hemos separado las componentes graficas de sus funcionalidades y los datos que representan, siguiendo el patrón modelo vista controlador (MVC). (Capitulo 7.2.1.2)

5.3.2 Tarjetas CRC.

A continuación definimos las tarjetas CRC que intervengan en el módulo de aprendizaje.

Tarjeta CRC

Nombre: DialogoNuevoPerfil

Funciones:
- Permite crear un nuevo perfil de usuario

Relación:
- VentanaAlumno

Tarjeta CRC

Nombre: MainFuncionamientoAlumno

Funciones:
- **C**rear la ventana del módulo de aprendizaje
- Realizar las llamadas a clases principales de los otros módulos que necesiten las clases de este paquete.

Relación:
- MainCBR
- MainInformacionDelSistema
- VentanaAlumno

Tarjeta CRC

Nombre: VentanaAlumno

Funciones:
- Contiene las funcionalidades del interfaz grafico del alumno (módulo de aprendizaje)
- Extiende VentanaAlumnoGUI

Relación:
- VentanaAlumnoGUI
- SelectorEjercicios
- DialogoNuevoPerfil
- MainFuncionamientoAlumno

Tarjeta CRC

Nombre: VentanaAlumnoGUI

Funciones:
- Contiene las componentes graficas necesarias para la resolución de ejercicios

Relación:
- VentanaAlumno

Tarjeta CRC

Nombre: SelectorEjercicios

Funciones:
- Permite elegir un ejercicio

Relación:
- VentanaAlumno

Tarjeta CRC

Nombre: Corrector

Funciones:

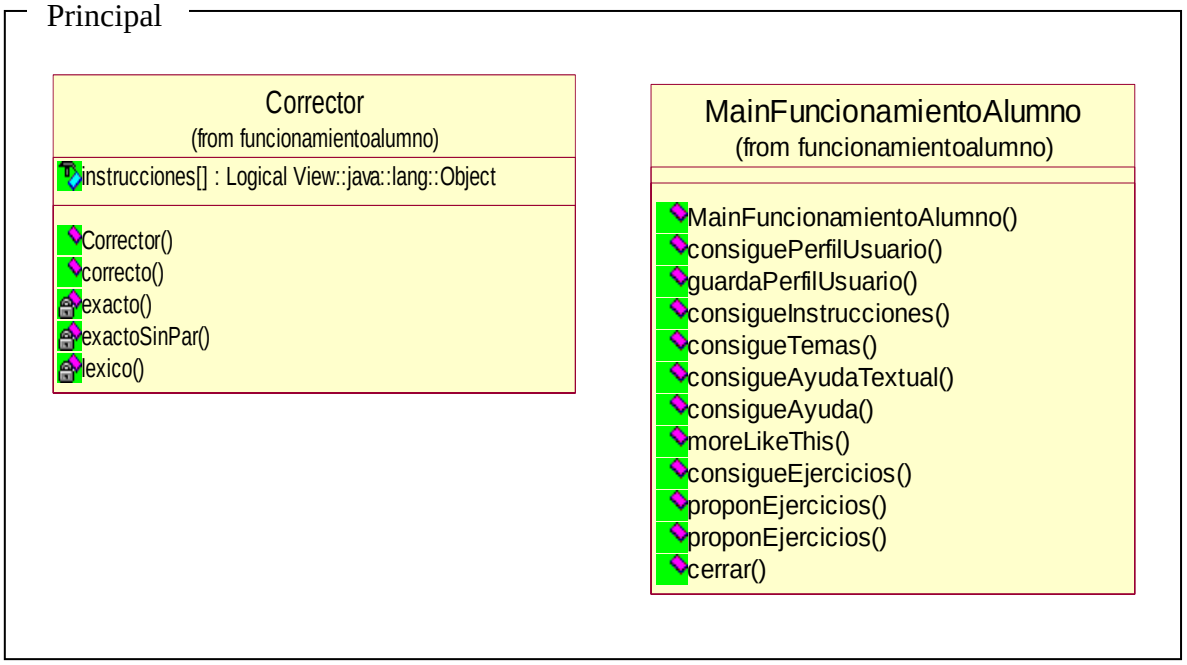
- Corregir las soluciones que plantea el alumno para los ejercicios propuestos

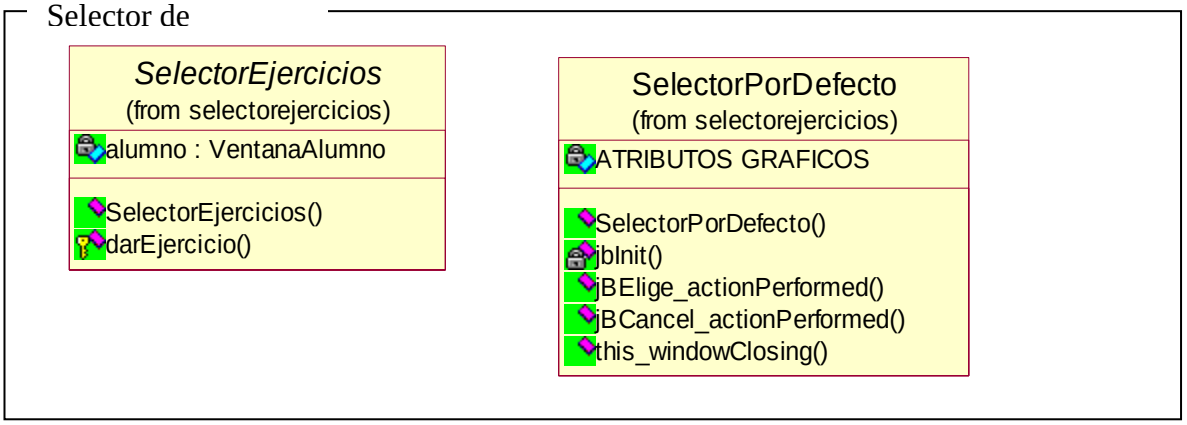
Relación:

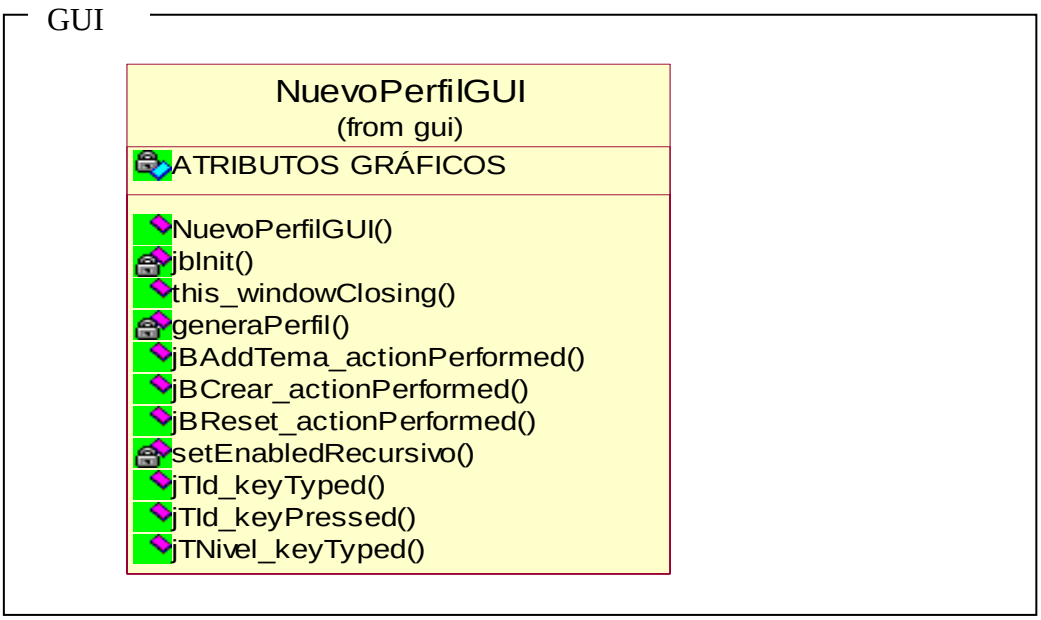
- VentanaAlumno

5.3.3 Diseño UML

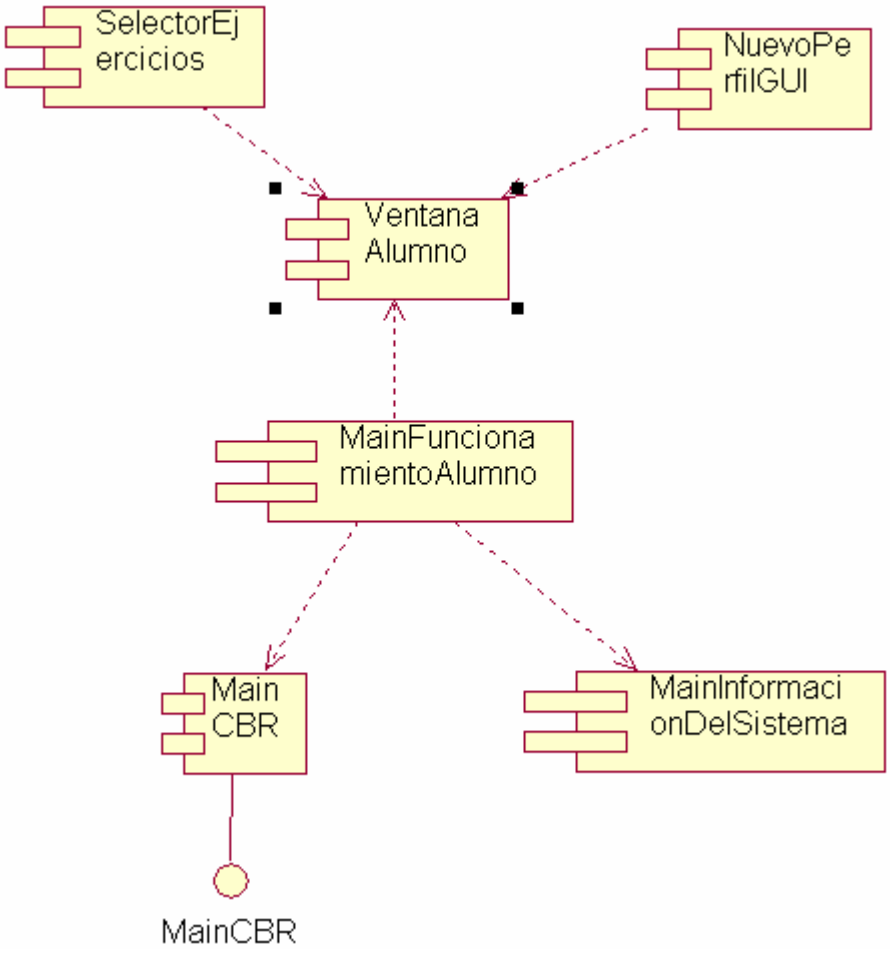
En primer lugar se muestra el diseño de las clases identificadas mediante las tarjetas CRC, y a continuación se mostrará el diseño correspondiente a la comunicación entre las clases del módulo de aprendizaje.







La comunicación entre las clases del módulo de aprendizaje se muestra a continuación:



5.4 SISTEMA CBR DE AYUDA CONTEXTUALIZADA

5.4.1 Introducción.

Utilizando la documentación de la fase de análisis, se comienza a diseñar la arquitectura del sistema CBR, para ello se decide crear una tarjeta CRC por cada una de las clases que van a formar el sistema, con ello se obtiene un diseño analítico que después se pasará a UML **[UML]**, y a partir de este diseño detallado en UML se generarán las clases.

5.4.2 Tarjetas CRC.

Explicamos a continuación las clases principales que intervienen en este módulo, y para cada una de ellas se indica que funciones tiene y con qué clases se comunican.

Tarjeta CRC

Nombre: MainCBR

Funciones:

- Comunicarse con el módulo de enseñanza y con el de creación de ejercicios.
- Añadir casos CBR.
- Dar ayuda al alumno.
- Dar ayudas similares a la anterior.
- Actualizar valoraciones de las ayudas.
- Devolver todos los casos y ayudas para su modificación o consulta.

Relación: Se relaciona con el módulo de enseñanza, el de creación de ejercicios y con las clases internas:

- Genera Ayuda
- AlmacenaCasos
- CBRTextual
- BaseDeCasos

Figura 5.4.1: Tarjeta CRC MainCBR

Tarjeta CRC

Nombre: BaseDeCasos

- **Funciones:** Es la base de Datos que contiene todo los casos CBR y las ayudas asociados a ellos:

Esta clase se limita a alojar toda la información necesaria para el sistema CBR y también en guardarla y cargarla de un XML.

Relación:

- Todas las clases del sistema CBR

Figura 5.4.2: Tarjeta CRC BaseDeCasos

Tarjeta CRC

Nombre: GeneradorAyuda

Funciones: Se limita a aplicar las funciones de similitud con todos los casos y a generar la ayuda

- Busca casos similares y devuelve un vector con los mejores

Relación:

- FuncionesSimilitud
- GeneradorAyuda

Figura 5.4.3: Tarjeta CRC GeneradorAyuda

Tarjeta CRC

Nombre: FuncionesSimilitud

Funciones: Se encarga de devolver la similitud entre dos casos CBR

- Tiene varias funciones de similitud:
  - Temas
  - Errores
  - Árboles
- Recibe dos casos y devuelve un valor de similitud entre ambos.

Relación:

- BuscadorCasos
- GeneradorAyuda

Figura 5.4.4: Tarjeta CRC FuncionesSimilitud

Tarjeta CRC

Nombre: BuscadorCasos

Funciones: Se encarga de buscar los casos similares con la mejor eficiencia posible.

- Busca casos similares
- Elige los mejores
- Genera la ayuda

Relación:

- MainCBR
- BuscadorCasos
- FuncionesSimilitud

Figura 5.4.5: Tarjeta CRC FuncionesSimilitud

5.4.3 Diseño UML.

El diseño UML que se desarrolló es básicamente lo que se había estudiado y pensado al crear las tarjetas CRC pero esta vez más de cara a la implementación para a partir de estos diagramas crear los módulos, paquetes y clases del proyecto.

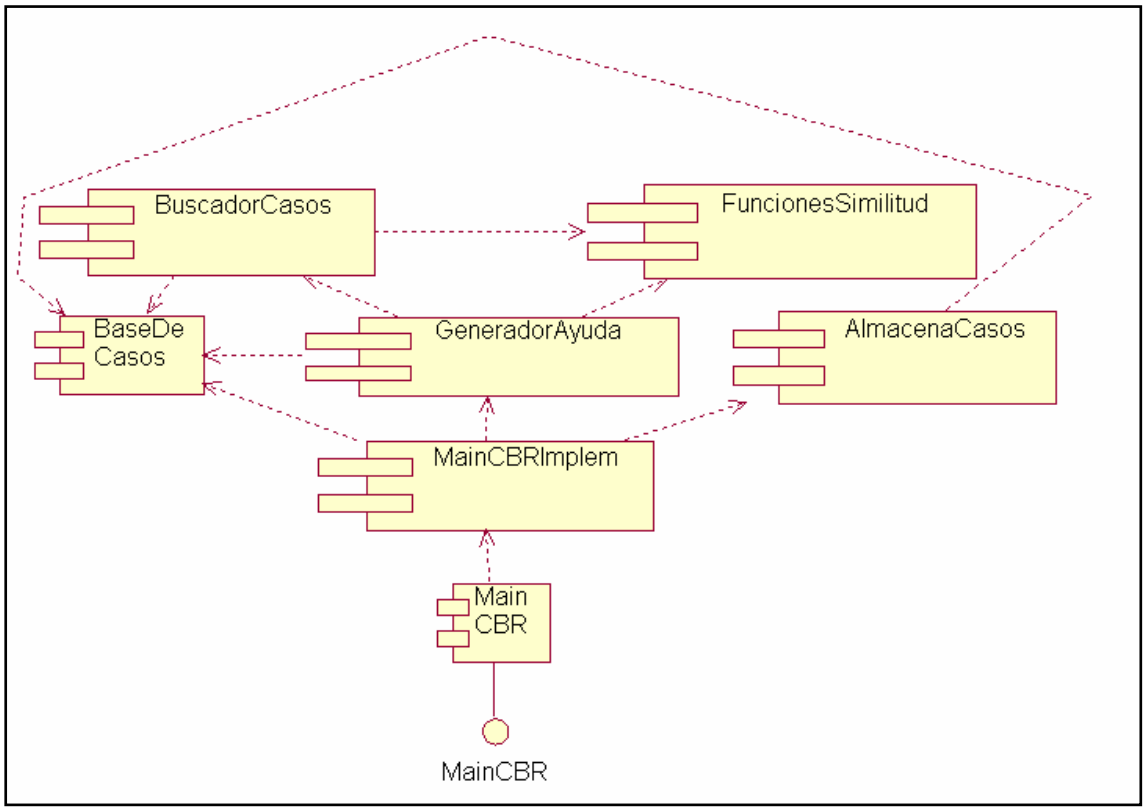


Figura 5.4.6: Diseño UML del paquete CBR.

Lo principal a destacar es la estructura de árbol con raíz en la clase 'MainCBR', la comunicación sigue una estructura jerárquica en forma de árbol de manera que las clases inferiores se comunican con la inmediatamente superior. Este esquema de diseño permite mayor modularidad, y por tanto mayor adaptabilidad al cambio.

Como se aprecia en la vista lógica (**Figura 5.4.7**) del diagrama de clases los métodos son los inferidos a partir de las tarjetas CRC.

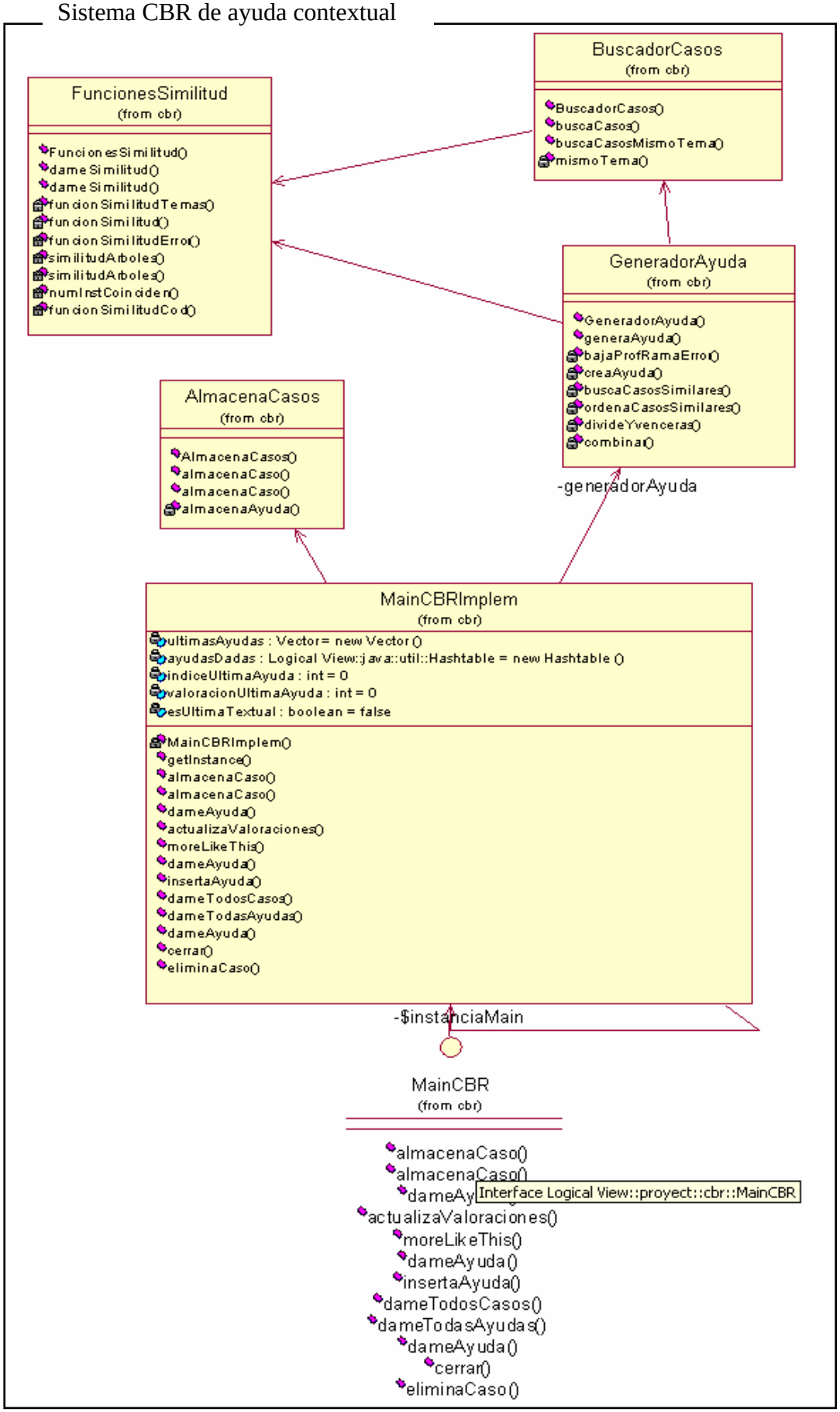


Figura 5.4.7: Diagrama UML vista lógica de las clases.

5.5 SISTEMA CBR DE AYUDA TEXTUAL

5.5.1 Introducción.

Para el diseño de esta parte del sistema se modificó parte de JColibri (**Capítulo 3.3.3.3**) y además se diseñó una serie de clases encargadas de intercomunicar tanto la parte creada con JColibri como con el resto de los módulos. A continuación detallamos el diseño de las dos clases que componen esta unión entre JColibri y el sistema de enseñanza.

5.5.2 Tarjetas CRC.

En este apartado se definen las clases que intervienen en este módulo, y para cada una de ellas se indica que funciones tiene y con qué clases se comunica.

Tarjeta CRC

Nombre: CBRTextual

Funciones:
Recibe una pregunta y devuelve la ayuda relacionada con una instrucción de la JVM.
O si es una pregunta interna es decir si el sistema CBR necesita información almacenada en la parte textual esta clase la devuelve.

Relación:
- MainCBR (al que puede acceder tanto el módulo de creación de ejercicios como el módulo de aprendizaje)
- GeneradorAyuda

Figura 5.5.1: Tarjeta CRC de la clase CBRTextual.

Tarjeta CRC

Nombre: textualCBR

Funciones: Contiene el sistema CBR Textual en sí. Contiene todos y cada uno de los métodos que se van a usar para entender y comparar los textos. Y es el encargado de ir ejecutando el preciclo, ciclo y postciclo, según se lo vaya pidiendo la clase CBRTextual.
Es la clase que interactúa con JColibri.

Relación:
- CBRTextual

Figura 5.5.2: Tarjeta CRC de la clase textualCBR.

5.5.3 Diseño UML.

A continuación se muestra el diseño UML con el diagrama de comunicación entre las clases descritas con las tarjetas CBR

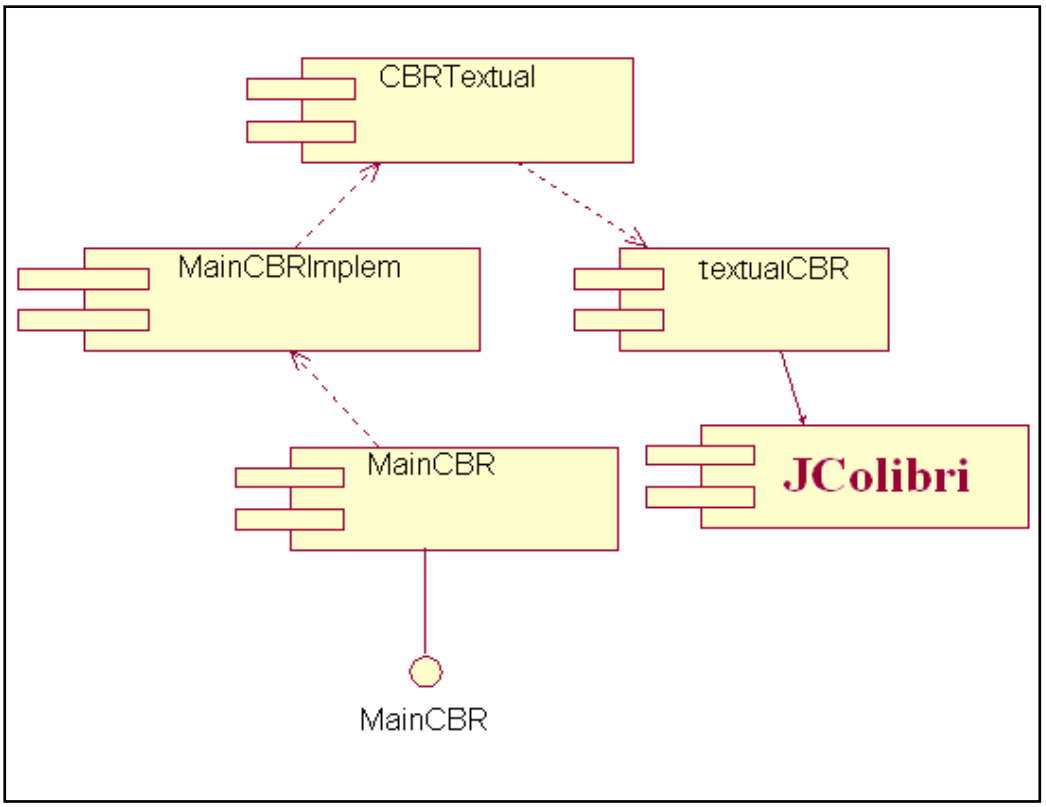


Figura 5.5.2: Diseño UML del paquete CBR textual.

La comunicación de este paquete es lineal ya que simplemente se necesita procesar la información con la clase CBRTextual y utilizar la clase textualCBR que es la clase resultado de utilizar el framework JColibri para crear la aplicación.

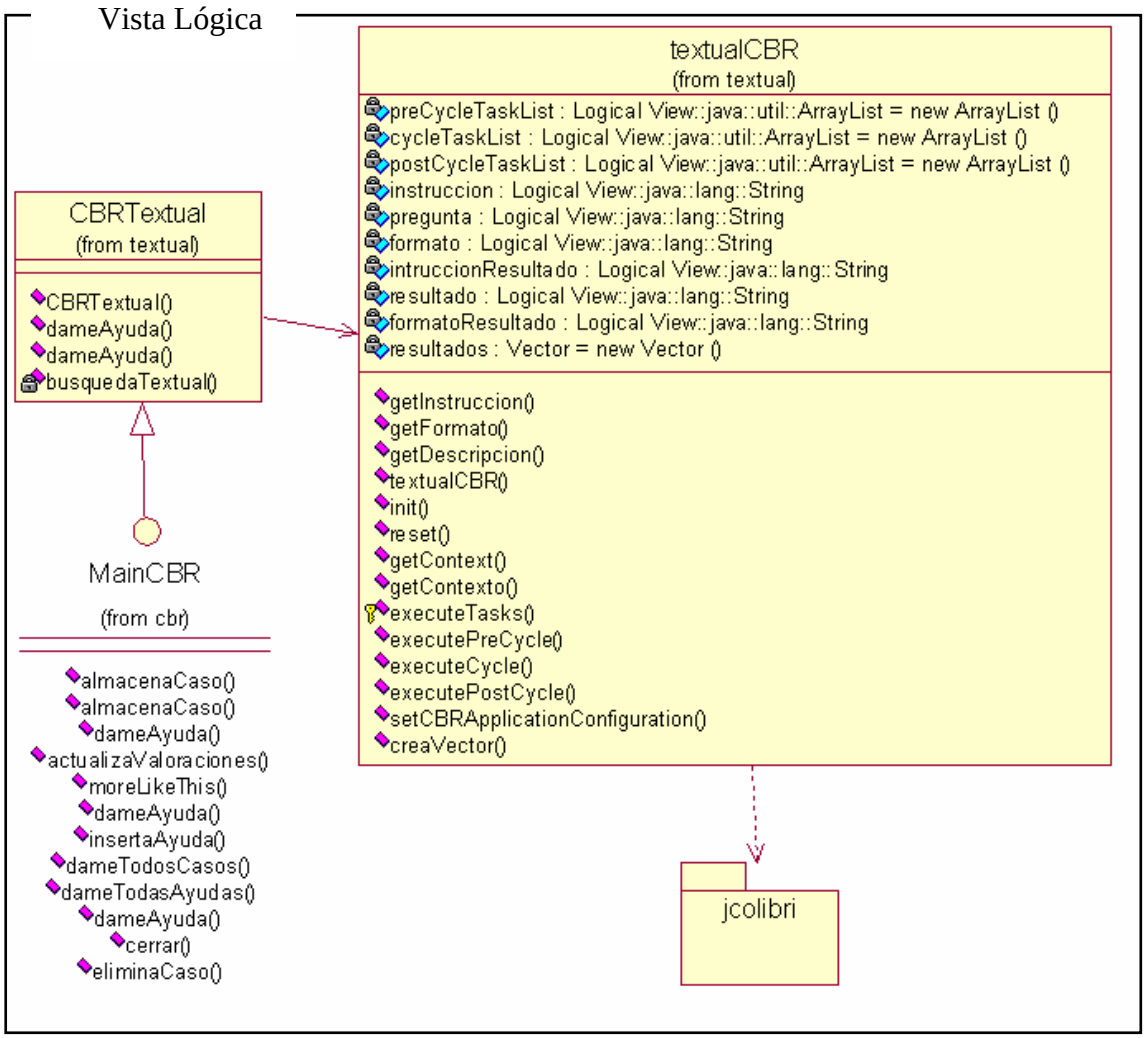


Figura 5.5.3: Diagrama UML vista lógica de las clases del sistema CBR de ayuda textual.

Se observar que la clase “CBRTextual” simplemente es un conector entre los módulos exteriores al CBR textual. Al igual que la clase textual CBR que es generada por JColibri al crear un sistema CBR configurando y creando diversos parámetros y clases Java.

5.6 MÓDULO DE INFORMACIÓN DEL SISTEMA

5.6.1 Introducción.

La idea es diseñar una clase principal, que se comunique con el resto de módulos. Esta clase no debe implementar directamente todas las funcionalidades que requiera este módulo, sino que delegará en otras clases internas dichas acciones. Tanto el diseño de la clase principal del módulo de información del sistema como las clases internas del mismo se detallan a continuación.

5.6.2 Tarjetas CRC.

En este apartado se definen las clases que intervienen en este módulo, y para cada una de ellas se indica qué funciones tiene y con qué clases se comunica.

Tarjeta CRC

Nombre: MainInformacionDelSistema

Funciones:

Permitir el acceso a toda la información del sistema, excepto la del CBR.Permitir la modificación, almacenamiento y eliminación de nueva información

Relación:

- MainFuncionamientoProfesor.
- MainFuncionamientoAlumno.
- InformacionJVM
- InformaciónEjercicios
- InformacionAlumno

Tarjeta CRC

Nombre: InformacionJVM

Funciones:

Almacenar en disco la lista con todas las instrucciones de la JVM.
Almacenar la lista de todos los temas de los que tratan las instrucciones.
Permitir el acceso a las instrucciones de la JVM
Permitir el acceso a los temas de las instrucciones de la JVM

Relación:

- MainInformacionDelSistema.
- ConectorJavaOwl

Tarjeta CRC

Nombre: InformacionEjercicios

Funciones:

Permitir el acceso a un ejercicio.
Permitir el acceso a la jerarquía de ejercicios.
Permitir el acceso a la lista de temas.
Almacenar toda la información correspondiente a los ejercicios.
Almacenar toda la información correspondiente a los temas.

Relación:

- MainInformacionDelSistema
- AccesoInformacionEjercicios.
- ModificadorInformacionEjercicios.

Tarjeta CRC

Nombre: AccesoInformacionEjercicios

Funciones:

Permitir el acceso a un ejercicio.
Permitir el acceso a la jerarquía de ejercicios.
Permitir el acceso a la lista de temas.

Relación:

- MainInformacionDelSistema
- AccesoInformacionEjercicios.
- ModificadorInformacionEjercicios.

Tarjeta CRC

Nombre: ModificadorInformacionEjercicios

Funciones:

Permitir la modificación de los ejercicios almacenados.
Permitir el almacenamiento y eliminación de ejercicios.
Permitir el. Almacenamiento de temas.

Relación:

- InformacionEjercicios.

Tarjeta CRC

Nombre: InformacionAlumnos

Funciones:

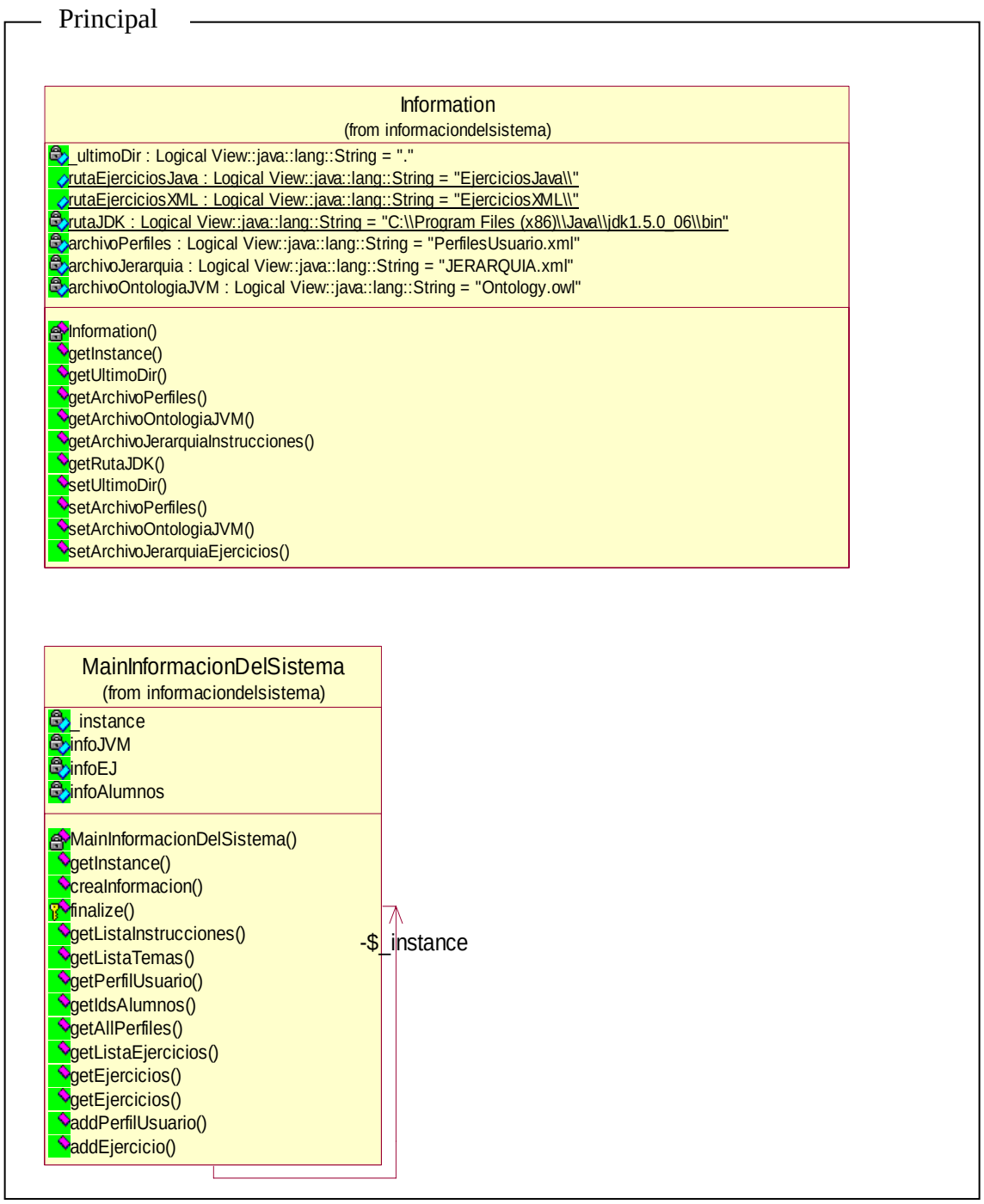
Permitir el acceso a los perfiles de usuario.
Almacenar un perfil de usuario.
Modificar un perfil de usuario.
Eliminar un perfil de usuario.

Relación:

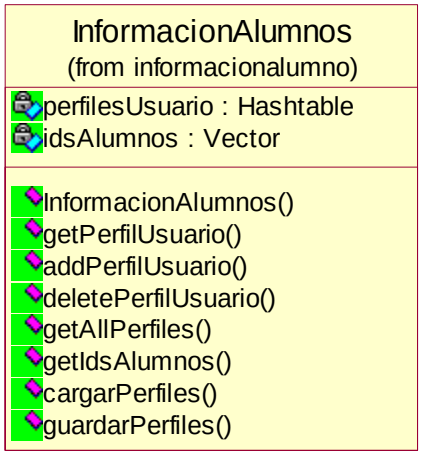
- MainInformacionDelSistema

5.6.3 Diseño UML.

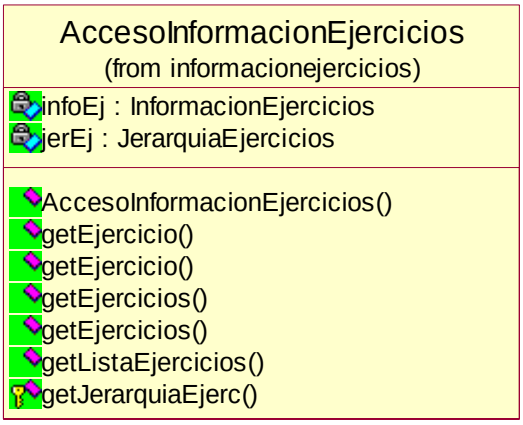
En este apartado se muestra el diseño para el módulo de información del sistema. En primer lugar se muestra el diseño para cada tarjeta CRC, y a continuación se mostrará es diseño de comunicación entre las clases que intervienen en el módulo de información.

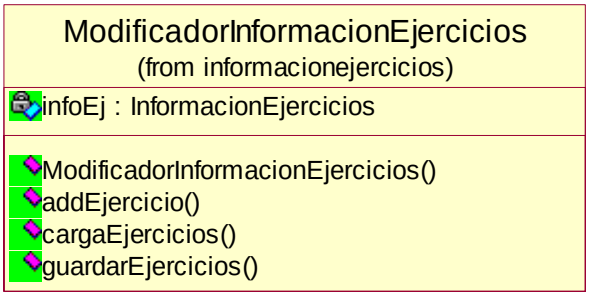


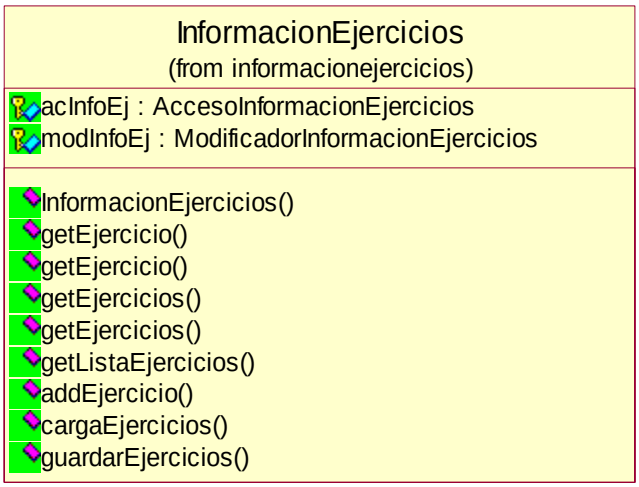
Información Alumno



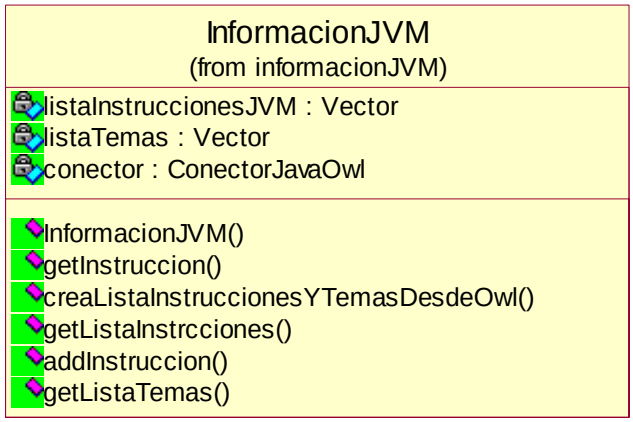
Información Ejercicios



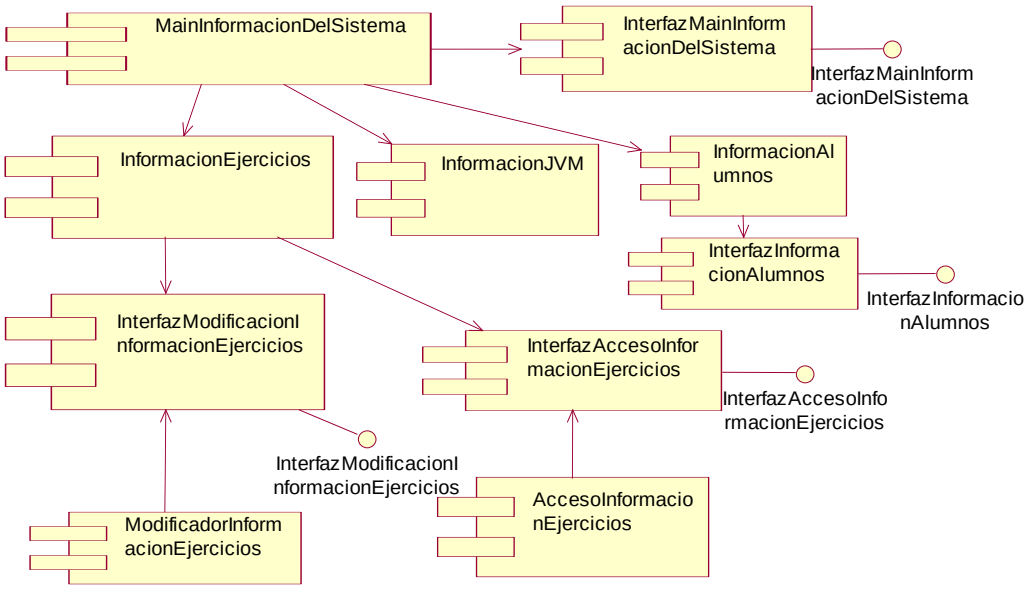




Información JVM



A continuación se muestra el diagrama de comunicación entre las clases que intervienen en el módulo de información del sistema.



Lo principal a destacar es la estructura de árbol con raíz en la clase 'MainInformacionDelSistema', esto es porque la comunicación sigue una estructura jerárquica en forma de árbol, las clases inferiores se comunican con la inmediatamente superior. Esta estructura de diseño permite mayor modularidad, y por tanto mayor adaptabilidad al cambio.

5.7 TIPOS

5.7.1 Introducción.

En este apartado se trata de diseñar las estructuras de datos (tipos) que van a representar la información con la que trabaja el sistema. El diseño de buenas estructuras de datos permite accesos rápidos a la información de dicha estructura.

Al igual que se ha hecho en los apartados anteriores primero identificaremos las estructuras de datos mediante las tarjetas CRC, y posteriormente se detallarán mediante su diseño.

5.7.2 Tarjetas CRC.

Tarjeta CRC

Nombre: PerfilUsuario

Funciones:

Representar la información acerca de un alumno.
Tiene que permitir saber el nivel de conocimiento del alumno.

Relación:

Tarjeta CRC

Nombre: InstruccionJVM

Funciones:

Representar la información extraída de las instrucciones de la ontología.

Relación:

Tarjeta CRC

Nombre: Ejercicio

Funciones:

Representar la información sobre un ejercicio.

Relación:

Tarjeta CRC

Nombre: JerarquiaEjercicios

Funciones:

Representar la información sobre la jerarquía de ejercicios. Esta información es para proponer ejercicios a los alumnos según esté establecido en esta jerarquía, teniendo en cuenta el nivel del alumno y los temas que ha tratado.

Relación:

Tarjeta CRC

Nombre: Ayuda

Funciones:

Representar la ayuda que se le proporciona a un alumno para resolver un ejercici.

Relación:

Tarjeta CRC

Nombre: Base de ayudas

Funciones:

Contener todas las ayudas que puede proporcionar el sistema.
Permitir acceso a las ayudas
Permitir modificación y eliminación de las ayudas.

Relación:

Tarjeta CRC

Nombre: CasoCBR

Funciones:

Representar un caso, es decir, un ejercicio, un fallo que se cometió y la ayuda que se prestó para la resolución de ese fallo.

Relación:

Tarjeta CRC

Nombre: BaseDeCasos

Funciones:

Contener todos los casos que posea el sistema.
Permitir acceso a lo casos.
Permitir modificación y eliminación de los casos.

Relación:

Tarjeta CRC

Nombre: ClassFile

Funciones:

Representa un fichero de código compilado, del que se lee byte a byte para obtener la representación lógica del fichero binario.

Relación:

Tarjeta CRC

Nombre: Codigo

Funciones:

Representar el código de un ejercicio, tanto el código compilado como el código sin compilar.

Relación:

Tarjeta CRC

Nombre: ExplanationTree

Funciones:

Representar en forma jerárquica (en árbol) el código compilado.
La representación jerárquica estará asociada al código fuente.

Relación:

Tarjeta CRC

Nombre: Method

Funciones:

Representar internamente un método descrito en lenguaje Java.

Relación:

Tarjeta CRC

Nombre: CodFuente

Funciones:

Representar el código fuente del ejercicio, es decir, código en lenguaje Java antes de compilarlo.

Relación:

Tarjeta CRC

Nombre: MethodTable

Funciones:

Representar en forma de tabla el conjunto de los métodos de un archivo de código compilado.

Relación:

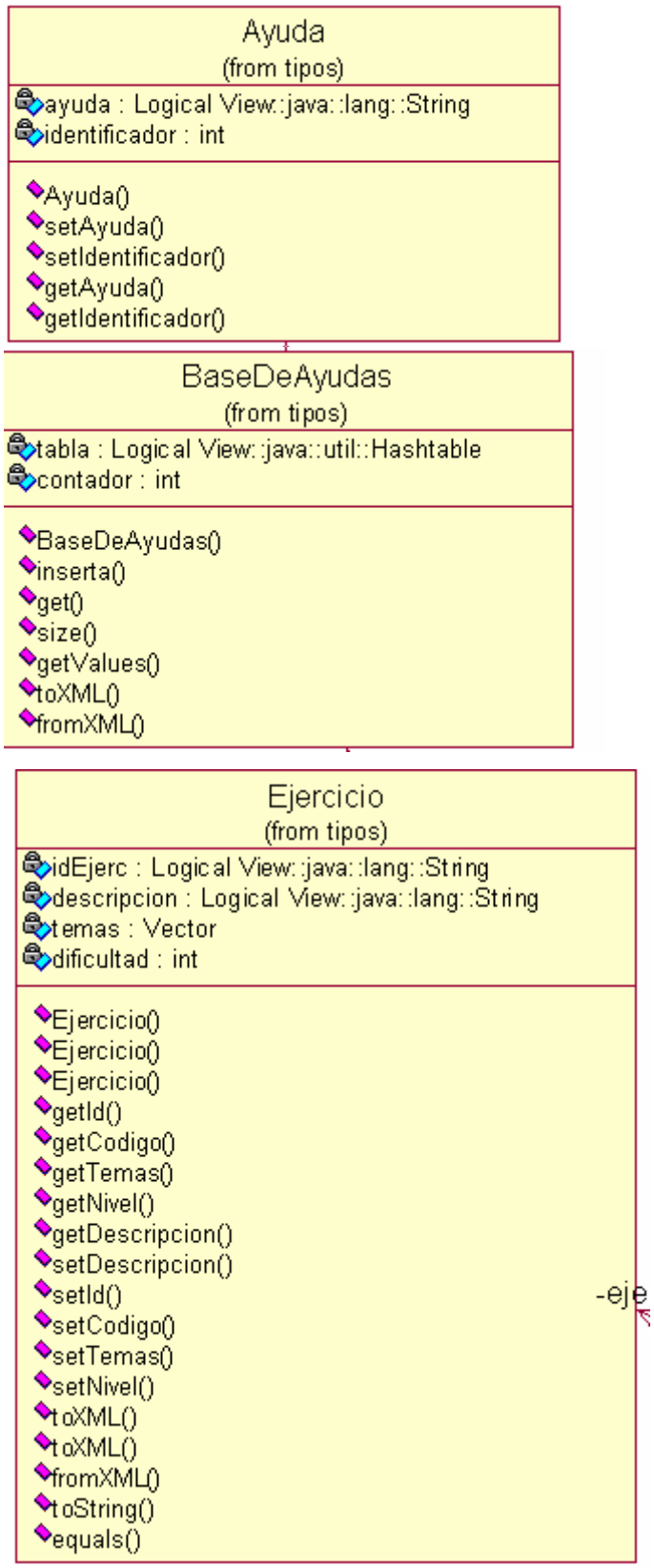
5.7.3 Diseño UML

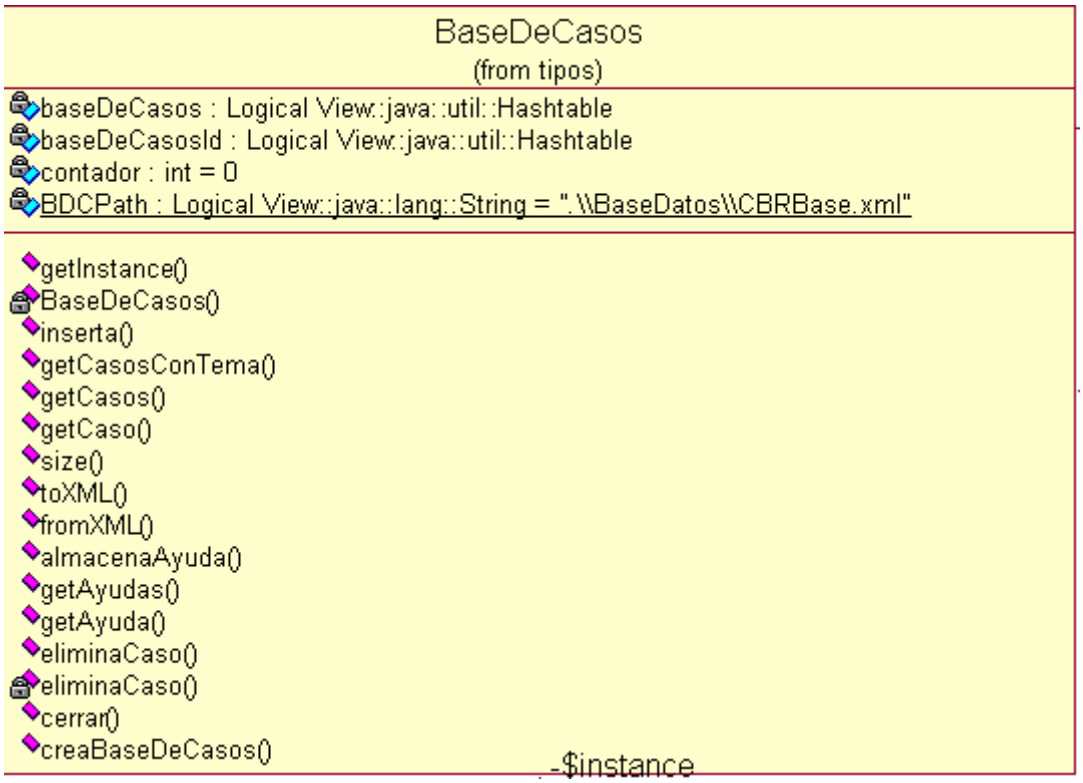
En un sistema software son muy importantes las estructuras de datos, un buen diseño de las mismas permitirá accesos, modificaciones y eliminaciones sobre los datos de forma eficiente.

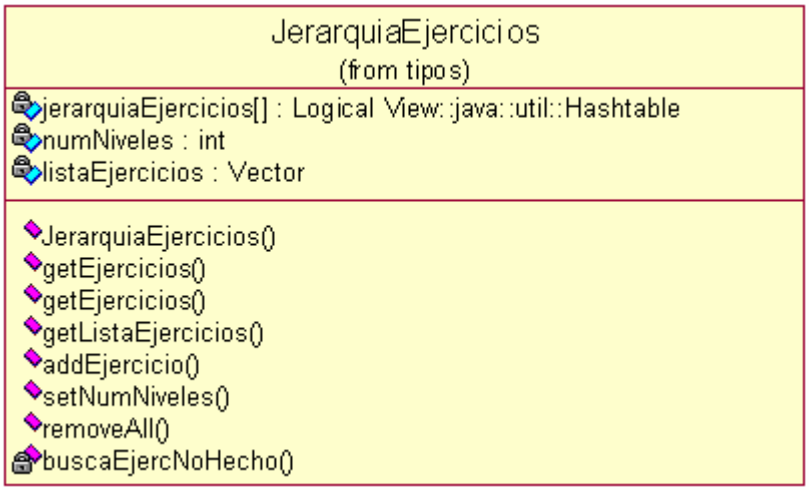
Por lo general, mejorar el acceso a una estructura de datos suele empeorar la modificación de los mismos, y viceversa. Con lo que se ha tenido en cuenta qué acciones se van a realizar más veces. Las bases de datos, en el sistema, la base de casos y de ayudas, requieren accesos rápidos, esto es para conseguir mayor rapidez en el funcionamiento de la aplicación de cara al usuario.

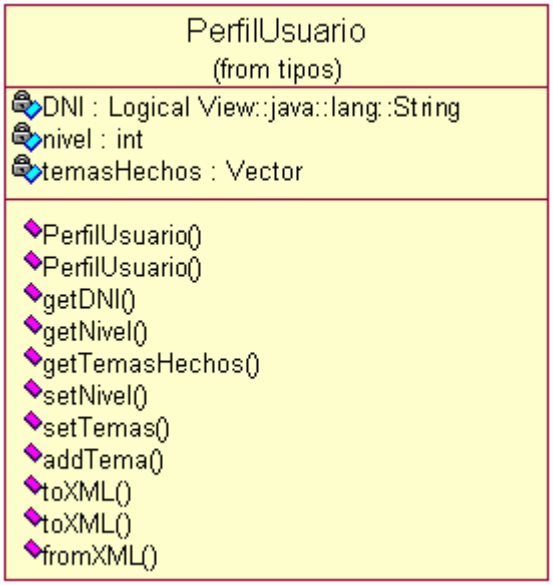
Por tanto en este apartado el trabajo se ha centrado en diseñar estructuras de datos eficientes.

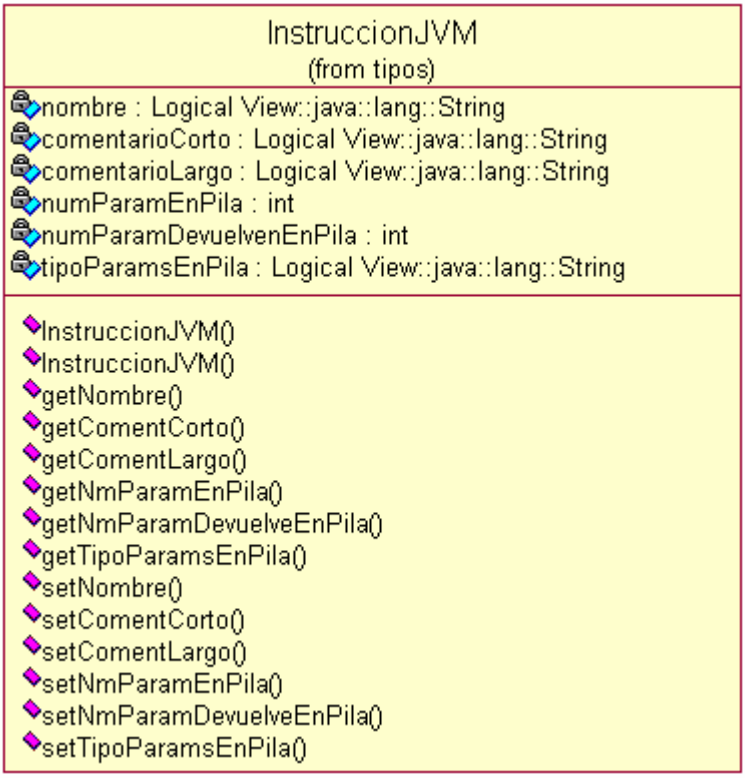
A continuación se muestra el diseño para cada tipo.

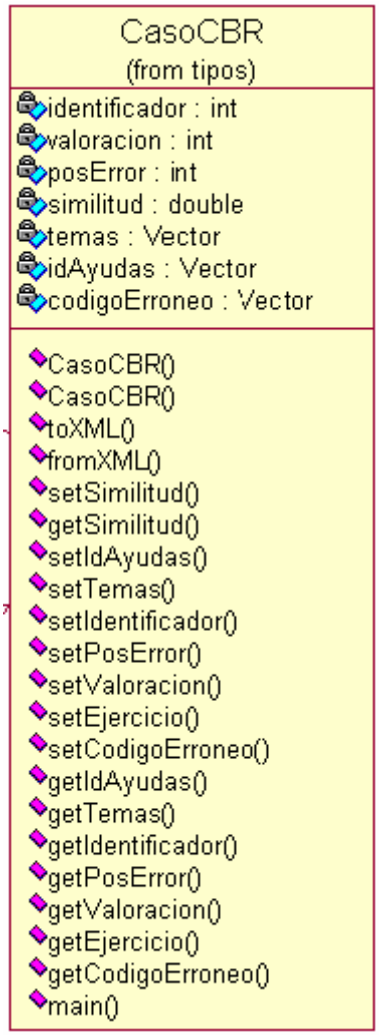

Ayuda
(from tipos)
ayuda : Logical View::java::lang::String
identificador : int
Ayuda()
setAyuda()
setIdentificador()
getAyuda()
getIdentificador()
BaseDeAyudas
(from tipos)
tabla : Logical View::java::util::Hashtable
contador : int
BaseDeAyudas()
inserta()
get()
size()
getValues()
toXML()
fromXML()
Ejercicio
(from tipos)
idEjerc : Logical View::java::lang::String
descripcion : Logical View::java::lang::String
temas : Vector
dificultad : int
Ejercicio()
Ejercicio()
Ejercicio()
getId()
getCodigo()
getTemas()
getNivel()
getDescripcion()
setDescripcion()
setId()
setCodigo()
setTemas()
setNivel()
toXML()
toXML()
fromXML()
toString()
equals()
-eje


BaseDeCasos
(from tipos)
baseDeCasos : Logical View::java::util::Hashtable
baseDeCasosId : Logical View::java::util::Hashtable
contador : int = 0
BDCPath : Logical View::java::lang::String = ".\\BaseDatos\\CBRBase.xml"
getInstance()
BaseDeCasos()
inserta()
getCasosConTema()
getCasos()
getCaso()
size()
toXML()
fromXML()
almacenaAyuda()
getAyudas()
getAyuda()
eliminaCaso()
eliminaCaso()
cerrar()
creaBaseDeCasos()
-$instance


JerarquiaEjercicios
(from tipos)
jerarquiaEjercicios[] : Logical View::java::util::Hashtable
numNiveles : int
listaEjercicios : Vector
JerarquiaEjercicios()
getEjercicios()
getEjercicios()
getListaEjercicios()
addEjercicio()
setNumNiveles()
removeAll()
buscaEjercNoHecho()


PerfilUsuario
(from tipos)
DNI : Logical View::java::lang::String
nivel : int
temasHechos : Vector
PerfilUsuario()
PerfilUsuario()
getDNI()
getNivel()
getTemasHechos()
setNivel()
setTemas()
addTema()
toXML()
toXML()
fromXML()


InstruccionJVM
(from tipos)
nombre : Logical View::java::lang::String
comentarioCorto : Logical View::java::lang::String
comentarioLargo : Logical View::java::lang::String
numParamEnPila : int
numParamDevuelvenEnPila : int
tipoParamsEnPila : Logical View::java::lang::String
InstruccionJVM()
InstruccionJVM()
getNombre()
getComentCorto()
getComentLargo()
getNmParamEnPila()
getNmParamDevuelveEnPila()
getTipoParamsEnPila()
setNombre()
setComentCorto()
setComentLargo()
setNmParamEnPila()
setNmParamDevuelveEnPila()
setTipoParamsEnPila()


CasoCBR
(from tipos)
identificador : int
valoracion : int
posError : int
similitud : double
temas : Vector
idAyudas : Vector
codigoErroneo : Vector
CasoCBR()
CasoCBR()
toXML()
fromXML()
setSimilitud()
getSimilitud()
setIdAyudas()
setTemas()
setIdentificador()
setPosError()
setValoracion()
setEjercicio()
setCodigoErroneo()
getIdAyudas()
getTemas()
getIdentificador()
getPosError()
getValoracion()
getEjercicio()
getCodigoErroneo()
main()

Tipo


ClassFile
(from tipocodigo)
name : Logical View::java::lang::String
superclassName : Logical View::java::lang::String
accessFlags : short
metodos : Vector
ClassFile()
ClassFile()
getMethods()
setMethodTable()
getAccessFlags()
getFieldTable()
getInterfaceTable()
setInterfaceTable()
setFieldTable()
setAccessFlags()
getName()
getSuperclassName()
setSuperclassName()
setName()
Codigo
(from tipocodigo)
nombre : Logical View::java::lang::String
explanationTree : ExplanationTree
codigoFuente : CodigoFuente
classFile : ClassFile
cargaDeArchivo()
nombreClase()
Codigo()
Codigo()
setNombre()
setCodigoFuente()
setExplanationTree()
setClassFile()
getNombre()
getCodigoFuente()
getExplanationTree()
getClassFile()
ExplanationTree
(from tipocodigo)
metodos : Vector
metodoSeleccionado : int
nombres : Vector
tipos : Vector
observers : Hashtable = new java.util.Hashtable ()
ExplanationTree()
ExplanationTree()
getProfundida()
getMetodos()
getNombres()
addMetodo()
addMetodo()
getTipos()
getTipo()
numMetodos()
getMetodoSelec()
getMetodo()
getMetodoSeleccionado()
setMetodoSeleccionado()
setMetodos()
getNombreMetodo()
getNombreMetodoSelec()
calcularLimiteSuperiorMetodo()
encontrarFinalMetodo()
addListener()
removeListener()
sendEvent()
toXML()
toXML()
fromXML()
Method
(from tipocodigo)
acces_flags : short
name : Logical View::java::lang::String
type : Logical View::java::lang::String
numExceptions : int
nameException[] : Logical View::java::lang::String
maxStack : int
maxLocals : int
lineinformationLenght : int
instructions : Vector
setInstructions()
getInstructions()
CodigoFuente
(from tipocodigo)
lineas : Vector
observers : Hashtable = new java.util.Hashtable ()
CodigoFuente()
CodigoFuente()
addLinea()
getLinea()
numLineas()
addListener()
removeListener()
sendEvent()
MethodTable
(from tipocodigo)
numMethods : int
MethodTable()

5.8 COMUNICACIÓN ENTRE LOS MÓDULOS

En este apartado se muestra el diseño UML de la comunicación entre los módulos diseñados en los apartados o capítulos anteriores. Desde el principio se decidió modular al máximo el proyecto para poder desarrollar e implementar los módulos por separado, es decir, que ninguno de los miembros del grupo tuviera que esperar a otro, a que terminará de implementar una parte, para poder continuar con su trabajo. Para ello creamos cuatro interfaces en los que se diseñaron los métodos que utilizarían para comunicarse con los demás módulos y por tanto todas las comunicaciones que se llevan a cabo entre los módulos que componen la aplicación se producen a través de estas cuatro clases, ninguna clase interna a un módulo se comunica con otro si no es a través de estos interfaces.

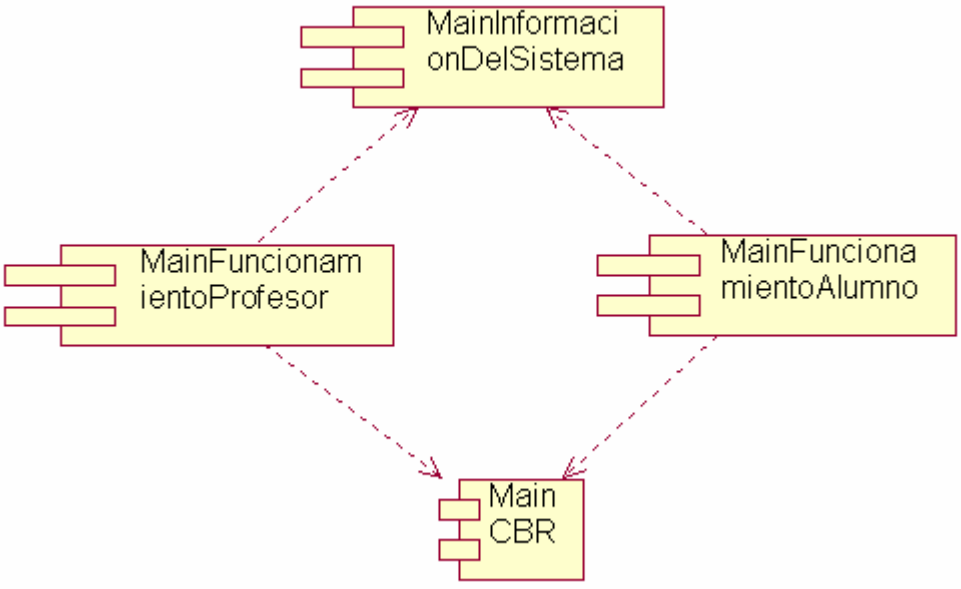


Figura 5.8.1: Comunicación entre los diferentes módulos.

CAPÍTULO 6
FASE DE DESARROLLO

6. FASE DE DESARROLLO

6.1 INTRODUCCIÓN

El objetivo de esta fase es desarrollar e instalar la aplicación empleando como guía el diseño del sistema, usando las herramientas estudiadas en la fase de investigación.

El código está desarrollado en lenguaje java, que cumple el requisito de una aplicación multiplataforma. Las interfaces gráficas se han desarrollado mediante el JBuilder 9.0, y el resto del código con el Eclipse SDK 3.1.2.

Gracias al diseño modular, los distintos módulos de la aplicación se han podido desarrollar en paralelo, permitiendo un mejor rendimiento por parte del grupo de desarrollo. El trabajo en paralelo es muy importante para el desarrollo de cualquier aplicación, que junto con una buena planificación permite un uso óptimo de los recursos. En particular para esta aplicación ha sido importantísimo, puesto que la incompatibilidad de horarios no permitía reunir al grupo de desarrollo tanto como fuera deseado.

6.2 SISTEMA CBR DE AYUDA TEXTUAL

6.2.1 Introducción

Ante la necesidad de ayudar al alumno a resolver los ejercicios nos planteamos la pregunta de cómo prestarle una ayuda fácil e intuitiva a cerca de las instrucciones de la JVM, de esta manera el alumno no tendría que saber de memoria todas y cada una de las instrucciones, o incluso podría describir el funcionamiento de una instrucción y recibir como resultado la instrucción que tiene esa funcionalidad.

Para solventar este problema y de nuevo con la ayuda de GAIA nos decantamos por la implementación de un sistema de CBR textual. Describimos a continuación las distintas partes del sistema.

6.2.2 Base de Casos.

En el **capitulo 4.2.2** ya se ha explicado cual es la estructura de los casos y de la base de casos, pero no se ha detallado la implementación de los mismos, así pues en este apartado se procede a detallar la estructura de la base de casos y los casos desde un punto de vista orientado a la implementación.

La representación en memoria que usamos para los casos CBR es un objeto CBRCaseRecord que tiene los siguientes atributos:

- *Nombre* instrucción.
- *Formato* de instrucción
- *Descripción* de la instrucción.

Estos casos son generados automáticamente por JColibrí [JColibri] con la ayuda del conector Java que hemos creado JConnector detallado en **capítulo 6.2.4** de implementación.

La base de casos es una lista enlazada (java.util.LinkedList) compuesta por objetos CBRCaseRecord. El LinkedList está pensado para realizar inserciones y

eliminación de objetos situados al principio de la lista. Es decir es la estructura adecuada para ir añadiendo casos al crear la base de casos y después poder recorrerla.

Ejemplo concerto:

```
java.util.LinkedList caselist;

CBRCaseRecord _case = createCase(caseStructure, _instruction,
                    _format,_description);

    caselist.add(_case);
```

Este es un fragmento de código de nuestra aplicación lo que hace es crear un caso y añadirlo a la lista enlazada.

6.2.3 Conocimiento procedimental

Como ya hemos explicado una de las partes más importantes de un sistema CBR es el cálculo de la similitud entre dos casos.

El sistema CBR textual utiliza la función de similitud del coseno, basada en el espacio vectorial.

Funcionamiento:

A partir de una serie de textos representativos Se obtienen las palabras distintas que aparecerán en los textos a tratar. Cada palabra distinta implica una dimensión del vector, el cual representará el texto. El valor de cada dimensión se obtiene mediante 2 factores:

-El número de apariciones de la palabra en el texto.

-El peso que se le da a la palabra.

Entre los vectores que representan los textos se calculará el coseno del ángulo que forman. Cuanto menor sea el ángulo más se aproxima el coseno a 1. Por tanto cuanto menor es el ángulo más se parecen las coordenadas de los vectores, es decir, más parecidos son los textos.

Ejemplo concreto:

Se quiere extraer un texto de la base de casos que trate lo más parecido al texto de consulta. Supongamos los textos traducidos a vectores con los pesos asociados a cada palabra del universo de discurso.

Consulta ejemplo:
V(12,34,5,8,13,1,0,0,0,3)
Base de casos:
V1(1,2,8,13,4,1,5,7,34,0)
V2(0,0,0,0,13,34,56,0,0,0)
V3(8,30,7,8,16,5,0,0,0,1)
V4(18,3,7,7,1,4,3,6,2,3)
V5(0,0,17,8,16,21,4,8,7,6)
V6(30,12,7,10,16,3,6,6,3,1)

simCoseno(V,V1) = 0.1815
simCoseno(V,V2) = 0.0768
simCoseno(V,V3) = 0.9803
simCoseno(V,V4) = 0.4884
simCoseno(V,V5) = 0.2869
simCoseno(V,V6) = 0.7059

Podemos comprobar que el texto más parecido a la consulta V, es el V3, ya que mirando sus pesos observamos que son los más parecidos a la consulta.

6.2.4 Implementación.

Para la implementación de este sistema se ha utilizado un framework para la construcción de sistemas CBR desarrollado en nuestra facultad (*La Facultad de Informática, UCM),* más concretamente por el "Group for Artificial Intelligence Applications" **[GAIA]** perteneciente al departamento de sistemas informáticos y programación. Este framework se llama ***JColibri*** **[JColibri]**.

Para la implantación del sistema hemos utilizado gran parte de JColibri, y una serie de clases que a continuación describimos:

- JVMConnector: encargada de extraer la información del archivo con el libro de JVM y crear los casos uno a uno hasta formar la base de casos.

- ResultFrame: es un frame para mostrar los resultados de la pregunta del usuario.

- textualCBR: es el .java que genera JColibri de la aplicación que se crea desde el interfaz principal. Contiene toda la información para poder tratar tanto los textos de la infamación de la JVM como los de la pregunta del usuario.

- CBRTextual: es la clase que se encarga de comunicar tanto la parte del alumno como las clases internas del paquete CBR con la aplicación de CBR Textual.

- JColibri: se han tenido que modificar partes importantes del código de JColibri para adaptarlo a las necesidades de nuestro proyecto.

Además para la aplicación se han creado dos archivos .xml que contiene la información de la estructura de casos y la información del conector:

- CaseStructure.xml: contiene los atributos que tiene un caso CBR (una instrucción con su información), el tipo y el valor de los mismos.

- Connector.xml: Contiene la situación del archivo para crear la base de casos, y la clase Java que se encarga de extraer y crear estos casos.

6.2.5 Validación y Verificación. Pruebas

El uso de CBR Textual es muy acertado para este tipo de problemas en las que hay que comparar dos textos y saber cuan parecidos son. Además el sistema podría ir aprendiendo nuevos casos para mejorar el rendimiento del mismo. Aunque en esta parte del sistema no tiene sentido que aprenda nuevos casos CBR ya que son instrucciones de la JVM.

Para la parte de pruebas hemos realizado varias preguntas con más o menos información, nuestro sistema se comporta bastante bien para la mayoría de las preguntas. Las peguntas más generales, que no aportan datos característicos de una instrucción concreta, dan, como es lógico, un resultado más general.

6.2.6 Ejemplos de funcionamiento del Sistema.

Antes de mostrar la ejecución del ejemplo es preciso explicar cómo funciona la aplicación de CBR Textual que hemos creado.

En primer lugar el sistema realiza el PRECICLO a continuación realiza CICLO y por último el POSTCICLO.

PRECICLO:

Se lee el archivo conector.xml y caseStructure.xml en el que se indica la estructura de cada caso.

En el conector.xml se indica la clase java que se utiliza, para leer el fichero con la información de la JVM, y el se indica también el archivo con la información de la JVM.

A continuación la clase JVMConnector va leyendo el archivo de la JVM y creando uno a uno los casos formando así la base de casos.

De estos casos se ejecutan varios métodos que hacen un filtrado de los textos, es decir se va quitando los artículos las preposiciones, después se miran adjetivos, sinónimos etc. De esta manera se va dejando los textos en una forma

esquemática, es una especie de resumen que consta de las palabras clave de las que tarta el texto y cada una con un peso diferente.

Los métodos son:

- WordsFilter
- Stemmer
- ExtractNames
- ExtractPhrases
- ExtractFeatures
- DomainTopicClassifier

CICLO:

En el ciclo se ejecutan los métodos anteriormente comentados pero esta vez sobre la pregunta del usuario. A continuación se recorre caso por caso la base de casos y se van aplicando las funciones de similitud entre el caso selecciona y el caso del usuario. Cuando ya se tiene la similitud con toda la base de casos se seleccionan los mejores casos y se devuelve un vector a la clase CBRTextual, la cual se encargará de proporcionárselo al módulo de aprendizaje.

POSTCICLO:

En esta parte se cierran descriptores de ficheros abiertos y demás.

Ejemplo de funcionamiento:

1) En primer lugar el usuario realiza una pregunta, o describe una instrucción.

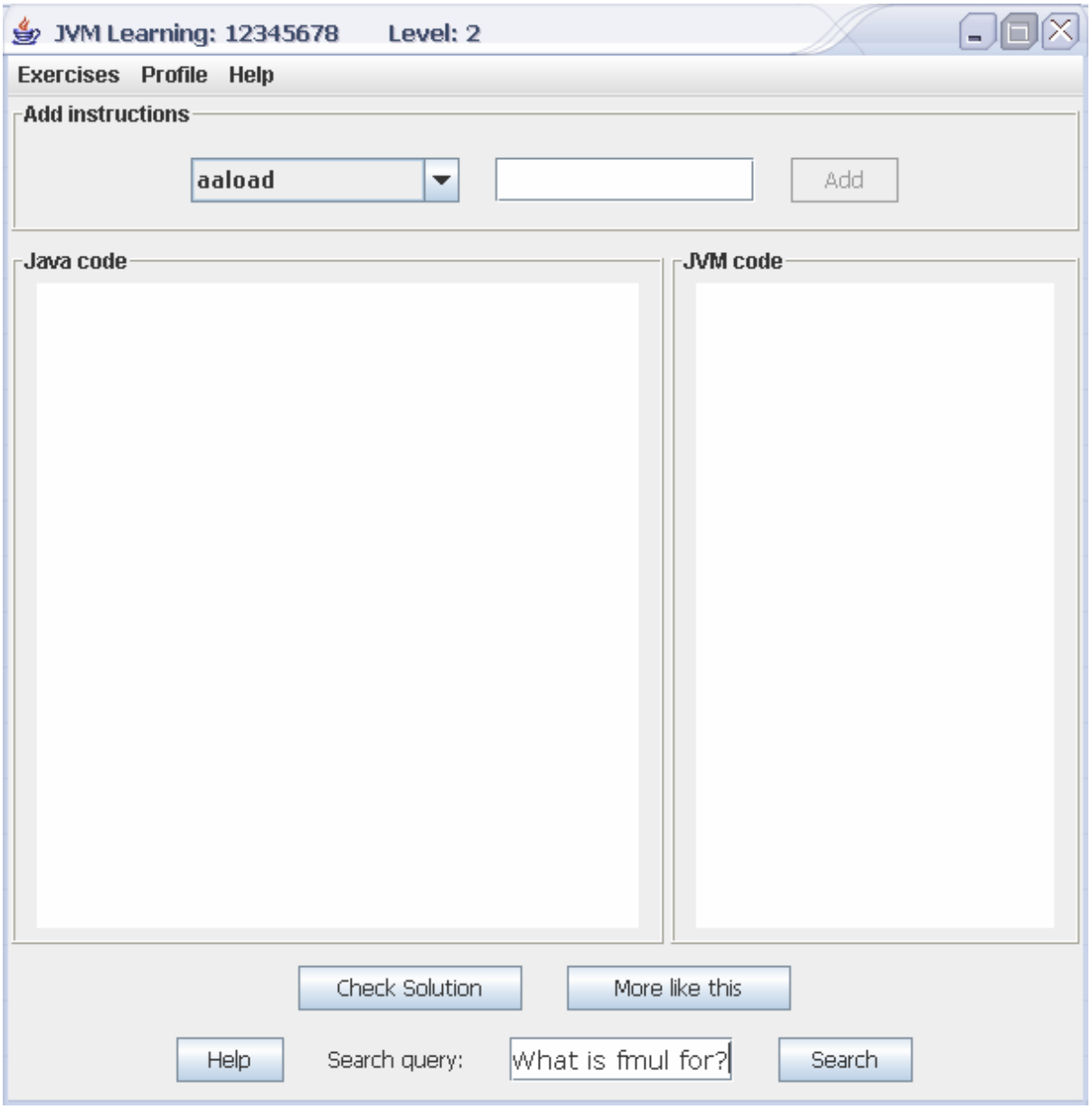


Figura 6.1: Uso del CBR textual (módulo de aprendizaje).

2) A continuación pulsa el botón "Search", el cual pone en marcha la aplicación de CBR Textual. Hace una llamada a CBRTextual pasándole la pregunta o descripción del usuario.

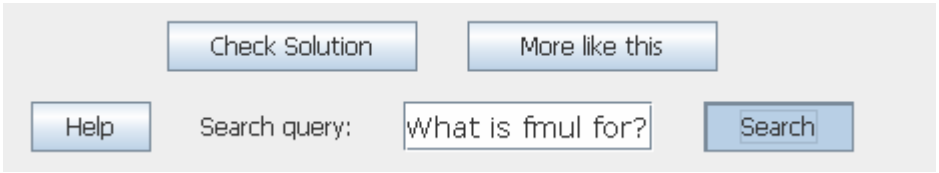


Figura 6.2: Detalle (módulo de aprendizaje).

3) El CBR Textual crea un objeto textualCBR lo inicializa con la pregunta del usuario y llama al preciclo. Ejecutando todos los métodos arriba enunciados.

Figura 6.3: Ejecución de los métodos de JColibri en la creacion de la base de casos.

4) A continuación llamamos a el ciclo, el sistema ejecuta los métodos de tratamiento textual a la pregunta para realizar la función de similitud del coseno con los demás casos.

Figura 6.4: Ejecución de los métodos y funcion de similitud de Jcolibri .

5) Para optimizar futuras consultas con el botón “More Like This”, creamos un vector con los mejores casos, es decir los que tiene mayor similitud. Así nos ahorramos tener que hacer muchas llamadas consecutivas al sistema CBR Textual.

6) Después el módulo del alumno nos muestra la ayuda solicitada.

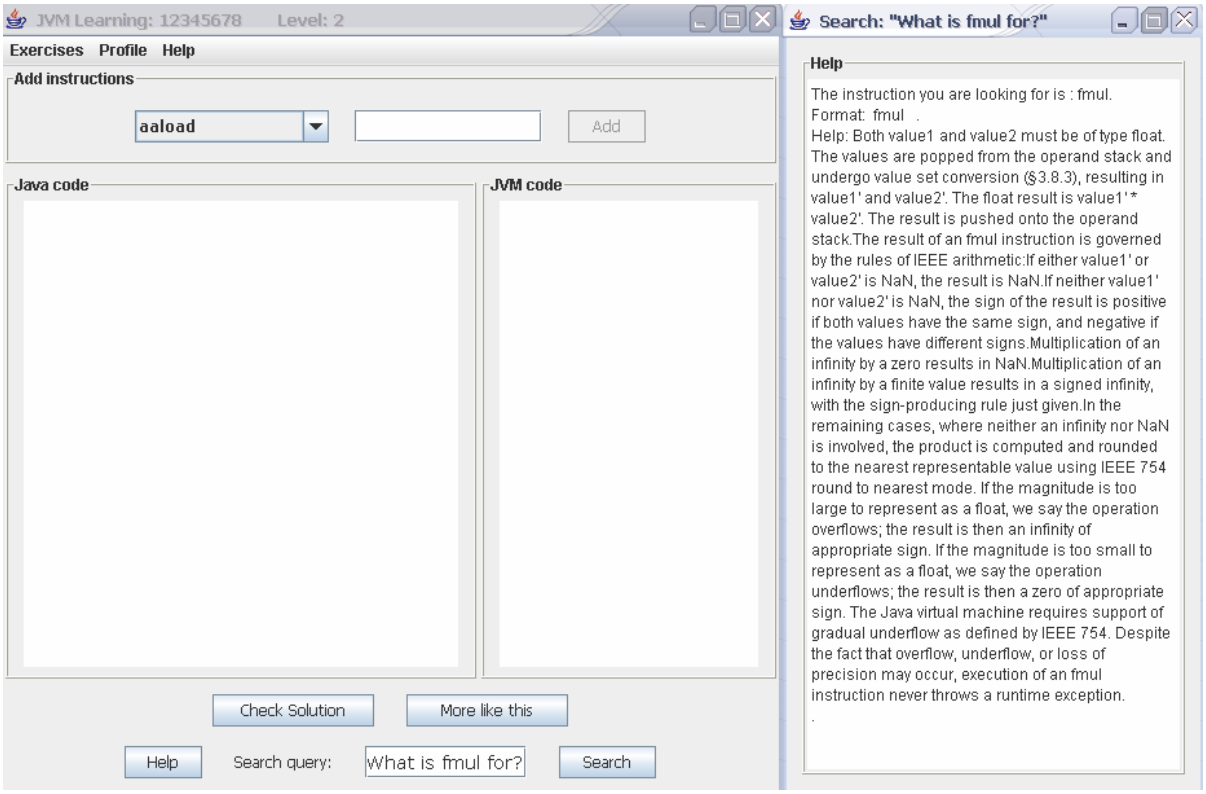


Figura 6.5: Módulo de aprendizaje mostrando la información solicitada.

7) Si se pulsa el botón “More Like This”, el sistema de forma inmediata nos da la siguiente instrucción con mayor similitud.

8) También se puede describir una instrucción sin hacer referencia al nombre de la misma. Por ejemplo: *which instruction is used to store a boolean into an array?* (**Figura 6.6**)

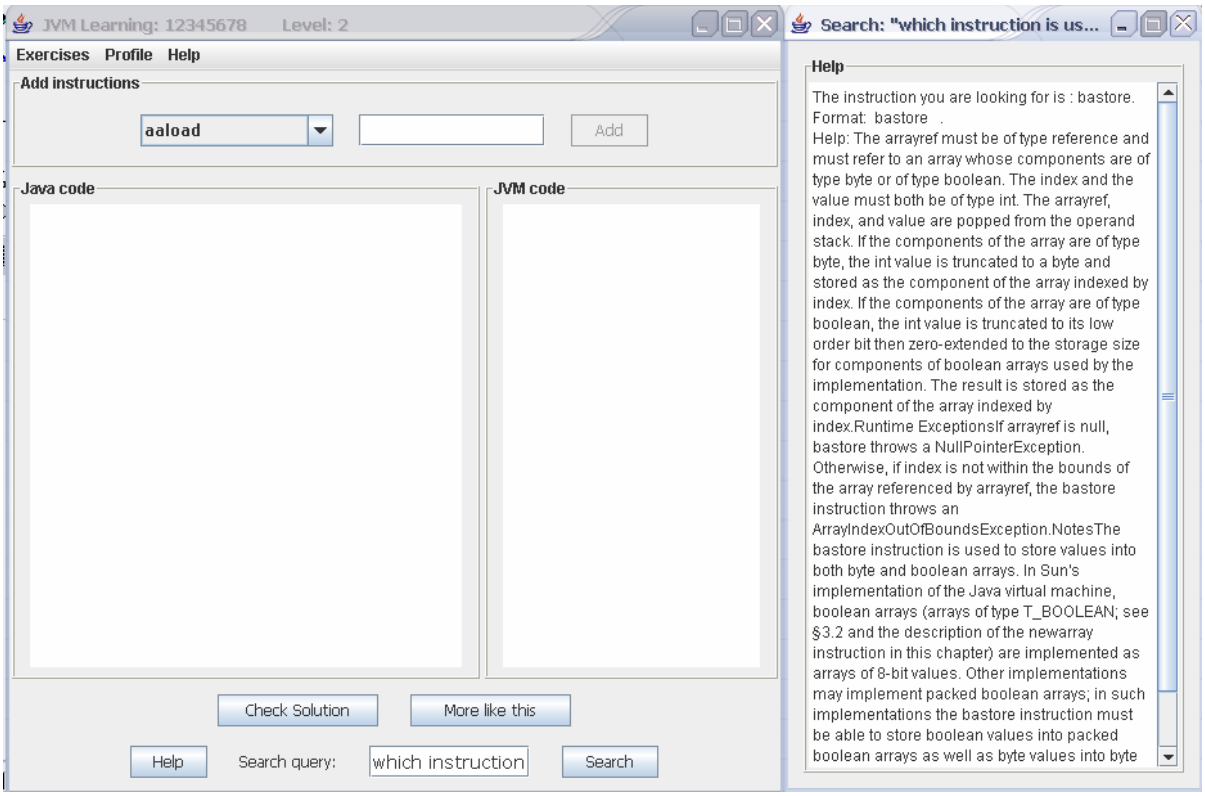


Figura 6.6: Módulo de aprendizaje mostrando la información solicitada con el botón “More Like This”.

6.2.7 Conclusiones.

El desarrollo de un sistema CBR textual no es tan sencillo como parecía, la implementación de un programa capaz de entender un texto.

La mayor dificultad son las diferentes pasadas que hay que hacer a un texto para dejarlo en su esqueleto, es decir para “resumir” el contenido del texto ,eliminando preposiciones, artículos, estableciendo similitud entre adjetivos, dando pesos a cada palabra…, aunque toda esta tarea se nos ha facilitado gracias a la librería de Java de WordNet (JWNL) y al framework JColibri.

Este sistema podría sacarse sin ningún problema de la aplicación he integrarlo en cualquier otro sistema. Es una aplicación muy útil a la hora de hacer preguntas o descripciones parciales de instrucciones de la JVM y podría servir como buscador refinado (Sistema CBR) no sólo para instrucciones sino para cualquier tipo de información, ya que el único cambio que habría que hacer

es en el conector JVMConnector.java es decir la forma de extraer la información del texto y por supuesto la base del conocimiento.

6.3 SISTEMA CBR DE AYUDA CONTEXTUALIZADA

6.3.1 Introducción.

Surge la necesidad de implementar el sistema CBR diseñado en las fases anteriores, así pues usando los diagramas de diseño UML(**Capítulo 5.4.3**) **[UML],** los casos de uso (**Capítulo 2.3**) y demás documentación comenzamos a implementar el sistema. En primer lugar se implementan las clases que forman el caso y la base de casos ya que son las partes primordiales de un sistema de razonamiento basado en casos. A continuación se implementan las distintas funciones de similitud y por último todas las clases y métodos de comunicación con los otros módulos de la aplicación.

6.3.2 Base de Casos

En el **capítulo 4.2.3** ya se ha explicado cuál es la estructura de los casos y de la base de casos, pero no se ha detallado la implementación de los mismos, así pues en este apartado pasamos a detallar la estructura de la base de casos y los casos desde un punto de vista orientado a la implementación.

La estructura de los casos es la siguiente:

- Identificador: que identifica inequívocamente a cada caso.
- Valoración: puntuación interna para la calificación de cada caso.
- PosError: posición del error cometido por el alumno o indicado por el profesor como error típico.
- Similitud: atributo interno para la elección del mejor caso.

- Temas: vector de temas relacionados con el ejercicio.

- IdAyudas: vector de identificadores de ayudas relacionadas con el error.

- Ejercicio: ejercicio con el código .java para proponer al alumno.

La estructura interna de la base de casos es muy compleja, ya que se ha enfatizado en la optimización temporal del sistema, es decir, se han utilizado estructuras de datos complejas para indexar los casos de manera que las búsquedas sean lo mas eficientes posibles en el dominio del tiempo. Por este motivo se han utilizado dos tablas Hash para guardar los casos, una de ellas indexada por los temas y otra por los identificadores. Además la base de casos consta de otra tabla Hash indexada por identificadores de ayudas.

El único problema con la complejidad de estas estructuras sólo se ve reflejado a la hora de insertar un caso o eliminarlo ya que se tiene que tener cuidado en mantener la consistencia y evitar la redundancia, por lo demás el comportamiento en las búsquedas es muy bueno.

6.3.3 Conocimiento procedimental.

Como en todo sistema CBR una de las partes fundamentales es el calculo de similitud entre dos casos, llamada función de similitud. Esta función como ya se ha explicado en el **capítulo 3.3.2** es la encargada de valorar cuan parecido son dos casos entre si. Las funciones de similitud que se utilizan en el sistema son bastante complejas ya que es difícil comparar dos casos tan complejos como los que posee el sistema.

En primer lugar se hace un primer filtrado en la base de casos seleccionando los casos que tiene algún tema en común con el caso que estamos tratando, a continuación se filtran nuevamente los casos fijándonos esta vez en los árboles que representan el ejercicio .java compilado para seleccionar los árboles con igual profundidad. A partir de este proceso obtenemos una base de

casos elitista y más reducida con la que se buscará caso por caso el caso con el error más parecido al cometido por el alumno en el caso que estamos tratando.

6.3.4 Implementación.

En la implementación de esta parte del proyecto se ha tenido siempre muy en cuenta la modularidad y la reutilización del sistema CBR, por ello esta compuesto por un número muy elevado de clases. Son las siguientes:

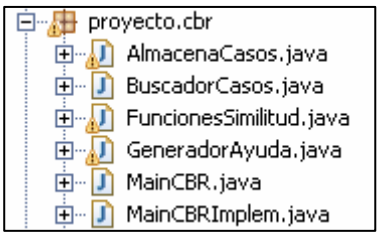


Figura 6.7:Estructura del paquete CBR

- AlmacenaCasos: es la encargada de almacenar los casos y ayudas.

- BuscadorCasos: se encarga de buscar casos similares al caso que entra como parámetro mediante la llamada ala clase funciones de similitud.

- FuncionesSimilitud: en esta clase están implementadas distintas funciones de similitud.

- GeneradorAyuda: se encarga de ordenar de mayor a menor mediante un algoritmo divide y vencerás los casos con mayor similitud devuelto por la clase buscador de casos y depuse compone las ayudas para que el módulo de aprendizaje las muestre al alumno.

- MainCBR: es la clase que se comunica con el resto de módulos e implementa métodos.

Además todas y cada una de estas clases utilizan los siguientes tipos:

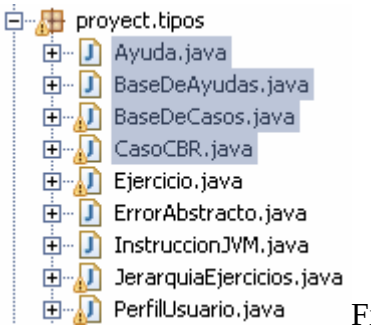
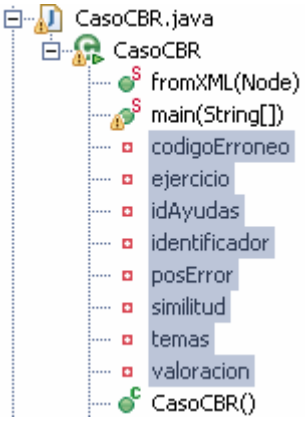


Figura 6.8: Tipos utilizados por el sistema CBR

- Ayuda: tipo que representa una ayuda contextual y tiene asignado un identificador único.

- BaseDeAyudas: es una tabla Hash indexada por los identificadores de las ayudas que contiene todas las ayudas.

- CasoCBR: Como ya se ha explicado en el capitulo X.X de adquisición del conocimiento las partes más importantes del caso CBR son el ejercicio propuesto por el profesor, el error cometido y la ayuda relacionada con el error cometido.



- Pero para la implementación se han añadido además varios atributos como un identificador para la identificar de manera inequívoca el caso, la posición de error para optimizar las búsquedas sobre el árbol que compone el código del ejercicio compilado, un vector de temas que indica que tipos de instrucciones de la JVM se necesitan para resolver el ejercicio, un número real que guarda la similitud con el caso con el que se esté comparando y por último la valoración de este caso en unidades de utilidad es decir 0 si no es un caso muy útil y 100 en caso contrario.

- BaseDeCasos: se tarta de una estructura para almacenar todos los casos CBR y ayudas. Para optimizar la eficiencia en las búsquedas hemos

implementado esta estructura con dos tablas Hash, como ya hemos explicado en el apartado anterior.

6.3.5 Validación y Verificación. Pruebas.

Las pruebas realizadas con esta parte de la aplicación han sido las mismas que las del módulo de enseñanza ya que dicho módulo utiliza el sistema CBR que estamos tratando como núcleo.

6.3.6 Conclusiones

Se ha conseguido la implementación de un sistema CBR totalmente modular y fácil de adaptar a cualquier tipo de conocimiento ya que solo habría que cambiar la estructura de los casos y de la base de casos. Y por supuesto implementar unas nuevas funciones de similitud correspondientes con esta nueva estructura de los casos y según las necesidades de la nueva aplicación.

El comportamiento del sistema es muy bueno en cuanto a eficiencia, tanto en tiempo como en calidad de resultados, la única pega que podría ponérsele es que la función de similitud que compara los árboles, que representan el ejercicio compilado, ralentiza un poco el sistema, pero la complejidad de dichos árboles no ha podido reducirse.

6.4 MÓDULO DE INFORMACIÓN DEL SISTEMA

6.4.1 Implementación

Este módulo no posee un interfaz gráfico, el acceso lo realiza de manera interna el sistema.

El proceso se ha llevado a cabo a partir del diseño, implementando cada método de cada clase que pertenece a este módulo. Puesto que la clase MainInformacionDelSistema sigue un patrón de diseño, concretamente el patrón singleton, la implementación ha sido acorde con dicho patrón.

La implementación no se ha llevado a cabo en una iteración, conforme se desarrollaban el resto de módulos surgían nuevas necesidades para el acceso y modificación de la información del sistema. Estas nuevas necesidades se traducen en la implementación de métodos nuevos, que a su vez se traduce en cambios en el diseño.

Los resultados tras la implementación no son visibles, excepto a nivel de código, que está detallado en el apéndice de este documento.

6.4.2 Validación y Verificación. Pruebas.

El proceso de probar el código implementado no se hace únicamente al terminar de desarrollar el módulo, sino que es un proceso integrado con la implementación.

Inicialmente se llevan a cabo las pruebas individuales de cada método, para pasar a comprobar el módulo por completo.

Puesto que los resultados no son visibles, no hay forma de ver gráficamente si los resultados son o no los esperados, con este objetivo se planteó un plan de pruebas para cada subsistema del módulo de información, que indica: el tipo de

prueba; el resultado esperado; comentarios cuando se llevó a cabo la prueba; y si está realizada o no.

1.-Información de los alumnos:

Tipo de Prueba	**Resultado esperado**	**Comentarios**	**Realizada**
Añadir correctamente un perfil de usuario	El perfil se almacene en la estructura que guarda los perfiles		
Añadir un perfil de usuario con DNI repetido	El sistema no debe hacer nada		
Acceder correctamente a un perfil de usuario	Devolver el perfil de usuario solicitado		
Acceder a un perfil de usuario no almacenado	El sistema lanzará una excepción indicando que no existe ningún perfil con el DNI solicitado		
Eliminar un perfil de usuario	El perfil de usuario debe ser eliminado correctamente		
Elimina un perfil de usuario que no está almacenado	El sistema no debe hacer nada		
Cargar los perfiles de usuario desde XML	El sistema debe almacenar de forma correcta (que puedan ser cargados) los perfiles de usuario en un fichero XML		
Guardar los perfiles de usuario en XML	El sistema debe cargar de forma correcta los perfiles de usuario desde un fichero XML		

2.- Información de la JVM:

Tipo de Prueba	Resultado esperado	Comentarios	Realizada
Crear la lista de temas y de instrucciones desde un fichero OWL	Almacenar de forma correcta la lista de temas y de instrucciones provenientes de la ontología de la JVM		
Acceder a una instrucción de la JVM	Devolver la instrucción con el nombre solicitado		
Acceder a una instrucción que no es de la JVM	El sistema debe lanzar una excepción indicando que no existe ninguna instrucción con el nombre solicitado		
Acceder a toda la lista de instrucciones	Devolver correctamente toda la lista de instrucciones		
Eliminar un perfil de usuario	El perfil de usuario debe ser eliminado correctamente		
Añadir una nueva instrucción	La instrucción debe ser añadida correctamente en la estructura correspondiente		
Añadir una instrucción ya almacenada	El sistema debe lanzar una excepción indicando que la instrucción y estaba almacenada		
Acceder a la lista de temas	Devolver correctamente toda la lista de temas		

3.- Información de ejercicios:

Tipo de Prueba	Resultado esperado	Comentarios	Realizada
Acceder a un ejercicio por nivel y tema	Devolver un ejercicio con ese nivel y tema, si no lo hay devolver null. Si el nivel es erróneo lanzar excepción		
Acceder a un ejercicio por nivel, tema, e indicando una lista de ejercicios que no quieres	Devolver un ejercicio con ese nivel y tema que no esté en la lista de ejerc que no quieres. Si no hay devolver null. Si el nivel es erróneo lanzar excepción		
Acceder a todos lo ejercicios en forma de lista	Devolver una lista con todos los ejercicios		
Acceder a la jerarquía de ejercicios	Devolver la jerarquía de ejercicios		
Añadir un ejercicio que no estaba	El sistema añade de forma correcta el ejercicio. Si el nivel es erróneo lanzar una excepción		
Añadir un ejercicio que estaba	El sistema no debe hacer nada		
Cargar la jerarquía de ejercicios desde XML	El sistema debe almacenar de forma correcta (que puedan ser cargados) la jerarquía de ejercicios en un fichero XML		
Guardar la jerarquía de ejercicios a XML	El sistema debe cargar de forma correcta la jerarquía de ejercicios desde un fichero XML		

Se puede observar que las dos últimas columnas están vacías, esto es porque el plan de pruebas se imprimió y se rellenó a mano conforme se sucedían las pruebas.

6.4.3 Conclusiones

Los conocimientos para la implementación de este módulo son sobre el lenguaje de programación y sobre patrones de diseño, concretamente el patrón singleton usado en la clase ‘MainInformacionDelSitema’.

La implementación del sistema no ha sido únicamente traducir el diseño, sino que al entrar en la implementación de otros módulos surgen nuevas necesidades que no habían sido previstas en el diseño.

6.5 EDITOR DE EJERCICIOS Y CASOS

6.5.1 Introducción

Este módulo es el encargado de interactuar con el profesor, dando herramientas graficas de fácil utilización para editar ejercicios y casos. A continuación detallamos por separado los dos interfaces y los detalles de su implementación.

6.5.2 Editor de ejercicios

Este módulo es el encargado de interactuar con el profesor, dando herramientas graficas de fácil utilización para editar ejercicios y casos. A continuación detallamos por separado los dos interfaces y los detalles de su implementación.

Esta es la parte del sistema donde el profesor crea y modifica los ejercicios que posteriormente va a realizar el alumno para adquirir los conocimientos deseados.

Un ejercicio se compone de lo siguientes campos:

- Un identificador, que es único y sirve para la organización interna de los ejercicios.

- Un nivel, que da una indicación sobre la dificultad del ejercicio.
- Una descripción, que sirve a modo de enunciado y detalla un poco más de que trata el ejercicio.
- Una lista de temas sobre los que trata el ejercicio, que es muy útil a la hora de proponer distintos ejercicios a distintos usuarios y a la hora de dar ayuda contextualizada a cada usuario.
- Un fragmento de código fuente, lo más importante del ejercicio, es el código que se propone para realizar la compilación. Concretamente en nuestro caso es en lenguaje Java. **[Java]**
- La compilación del código fuente, para comprobar la corrección de la solución que den los alumnos. En nuestro caso es en lenguaje JVM. **[JVM]**

El editor proporciona componentes visuales para modificar fácilmente los campos del ejercicio, el único problema que se plantea es la obtención del código compilado, que se realiza mediante código reutilizado del proyecto Jade, que llama al compilador del JDK e interpreta el archivo binario y lo carga en una estructura de datos conveniente para su manipulación**..** Los ejercicios se guardan en disco en formato XML y el código fuente en Java por separado en un archivo de texto normal.

6.5.3 Editor de casos

Esta es la parte del sistema donde el profesor crea, modifica y elimina los casos CBR que posteriormente van a servir para dar ayuda contextualizada a los alumnos que usen el sistema. En este sistema un caso es básicamente la asociación entre un ejercicio, un intento de solución (un error) y un conjunto de ayudas. **(Capítulo 4.2.3)**

Un caso tiene los siguientes campos básicos:

- Un ejercicio, que contiene la información de que ejercicio se estaba solucionando cuando se genero el caso.
- Un código compilado erróneo, que contiene el error concreto que se realizo.

- Una lista de ayudas que fueron dadas al alumno que realizaba el ejercicio.
- Una valoración, que da una idea de lo útil que han sido las ayudas de este caso en otras ocasiones.

El editor de casos es una extensión del editor de ejercicios con ventanas separadas para modificar el código erróneo, las ayudas asociadas al caso y la valoración del mismo. A diferencia que el editor de ejercicios los casos y las ayudas nuevas no se guardan en disco directamente sino que se actualiza la estructura que mantiene el sistema CBR en memoria y cuando se cierra el editor se vuelca a disco la base de casos completa. **(Capítulo 4.2.3)**

6.5.4 Implementación

La implementación se ha realizado con las herramientas de desarrollo de GUIs que proporciona el JBuilder 9.0, siempre siguiendo el diseño detallado en **(Capítulo 5.2)**, siguiendo siempre el patrón Modelo Vista Controlador detallado en el capitulo **(Capítulo 7.2.1.2)**

6.5.5 Validación y Verificación. Pruebas

Ambos editores se han probado intensivamente por separado y conjuntamente con todo el sistema para comprobar que cumplen las especificaciones y los diseños planteados.

Las pruebas conjuntas han servido para comprobar las conexiones con los módulos externos en este caso, la conexión con el módulo de información del sistema para la creación, el acceso y la manipulación de ejercicios y la conexión con el módulo CBR de ayuda contextualizada para la creación, el acceso y la manipulación de ayudas y casos CBR.

6.5.6 Ejemplos de funcionamiento del Sistema

A continuación se expone un ejemplo de funcionamiento del editor de ejercicios y casos:

1) Al iniciar el editor de ejercicios y casos se abre por defecto en modo editor de ejercicios, pudiéndose cargar ejercicios ya creados o crear nuevos.

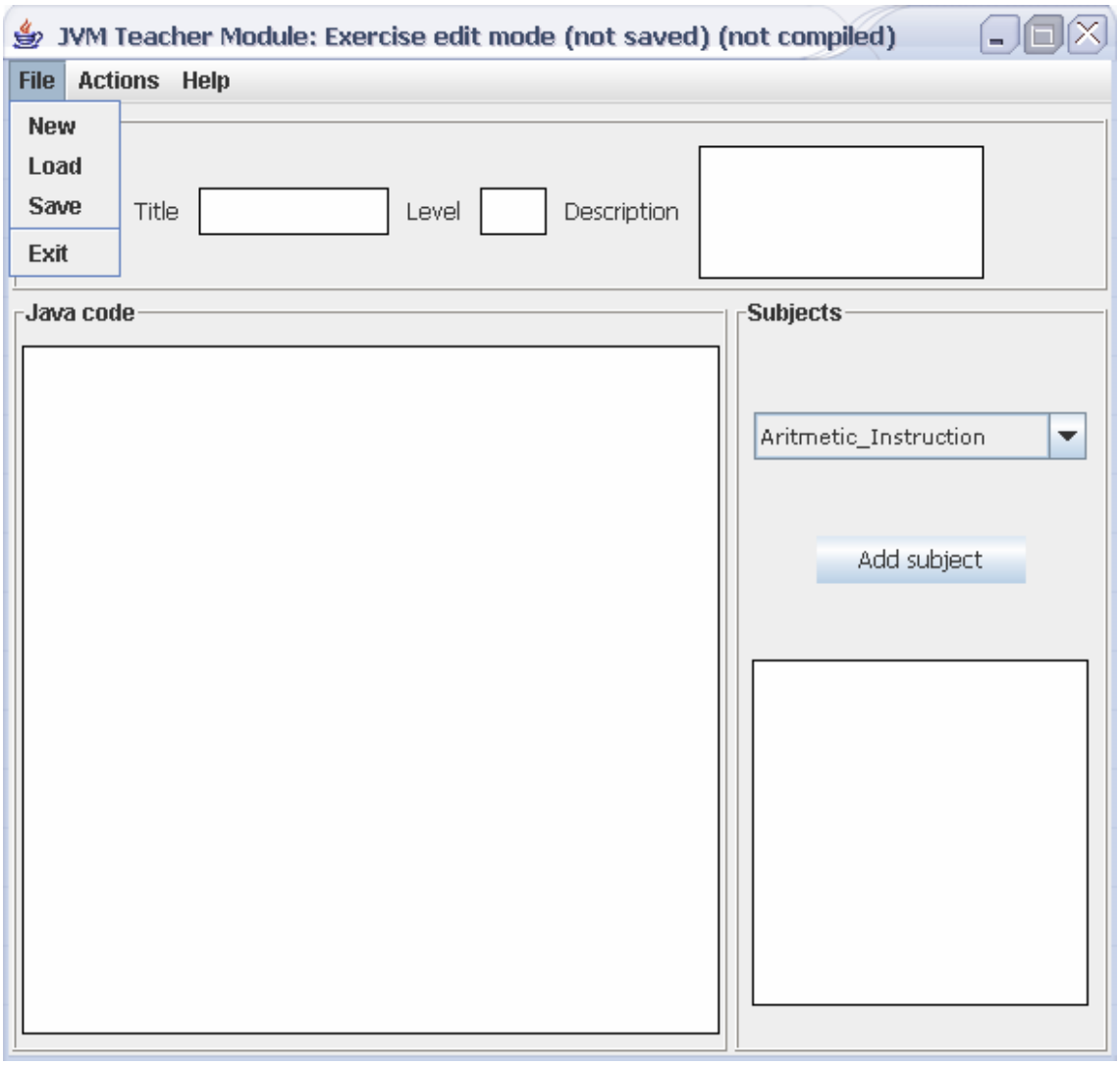


Figura 6.5.1: Editor en modo ejercicios.

2) A continuación se abre un ejercicio ya creado pulsando File → Load. Se abre un selector de fichero, aquí podemos cargar o bien un archivo xml creado por la aplicación o un archivo .java para la creación del ejercicio.

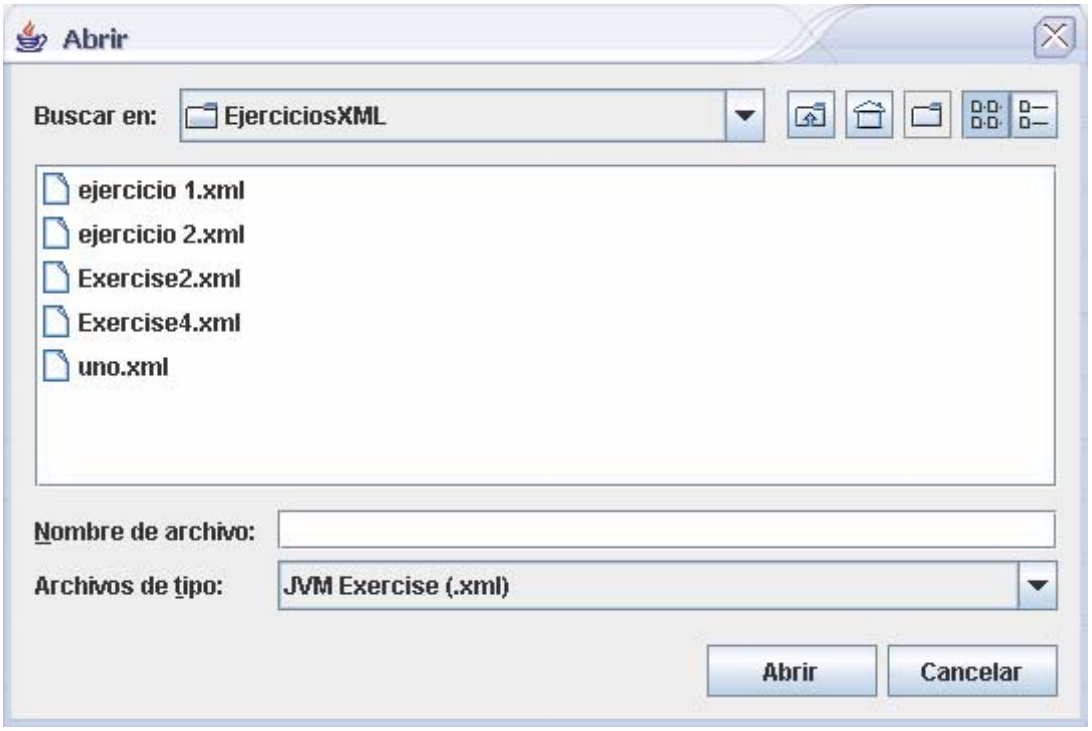


Figura 6.5.2: Selector de fichero para ejercicios en XML creados por la aplicación.

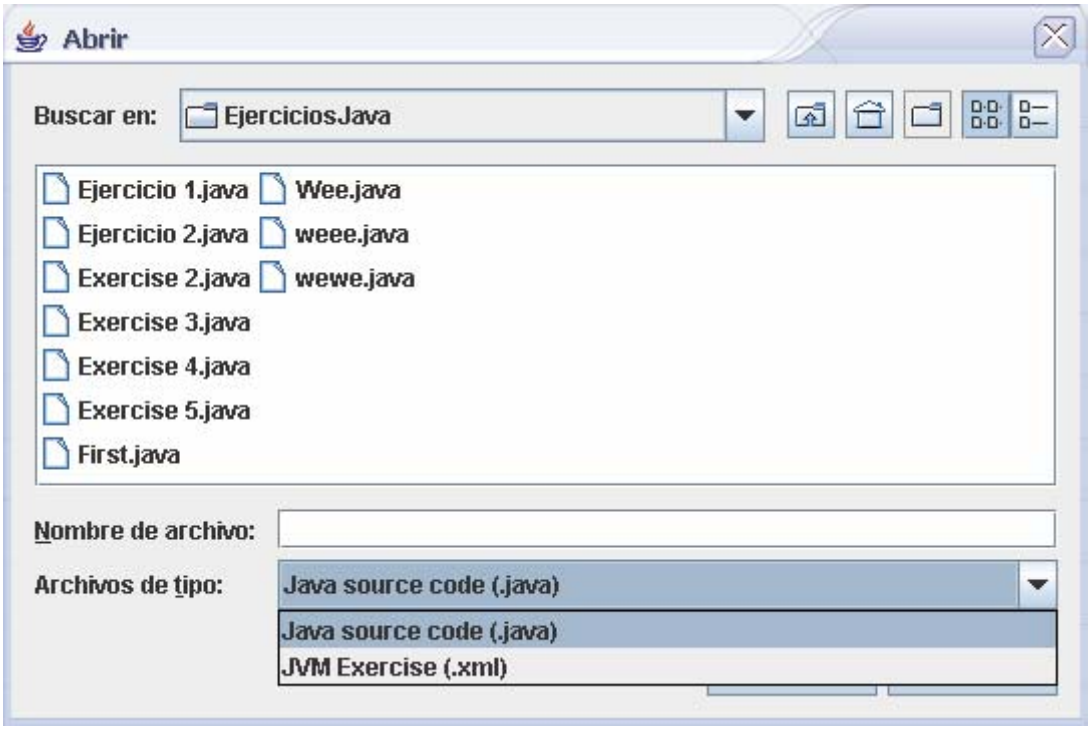


Figura 6.5.3: Selector de fichero para ejercicios java.

3) Al cargar el ejercicio este se hace visible en el editor.

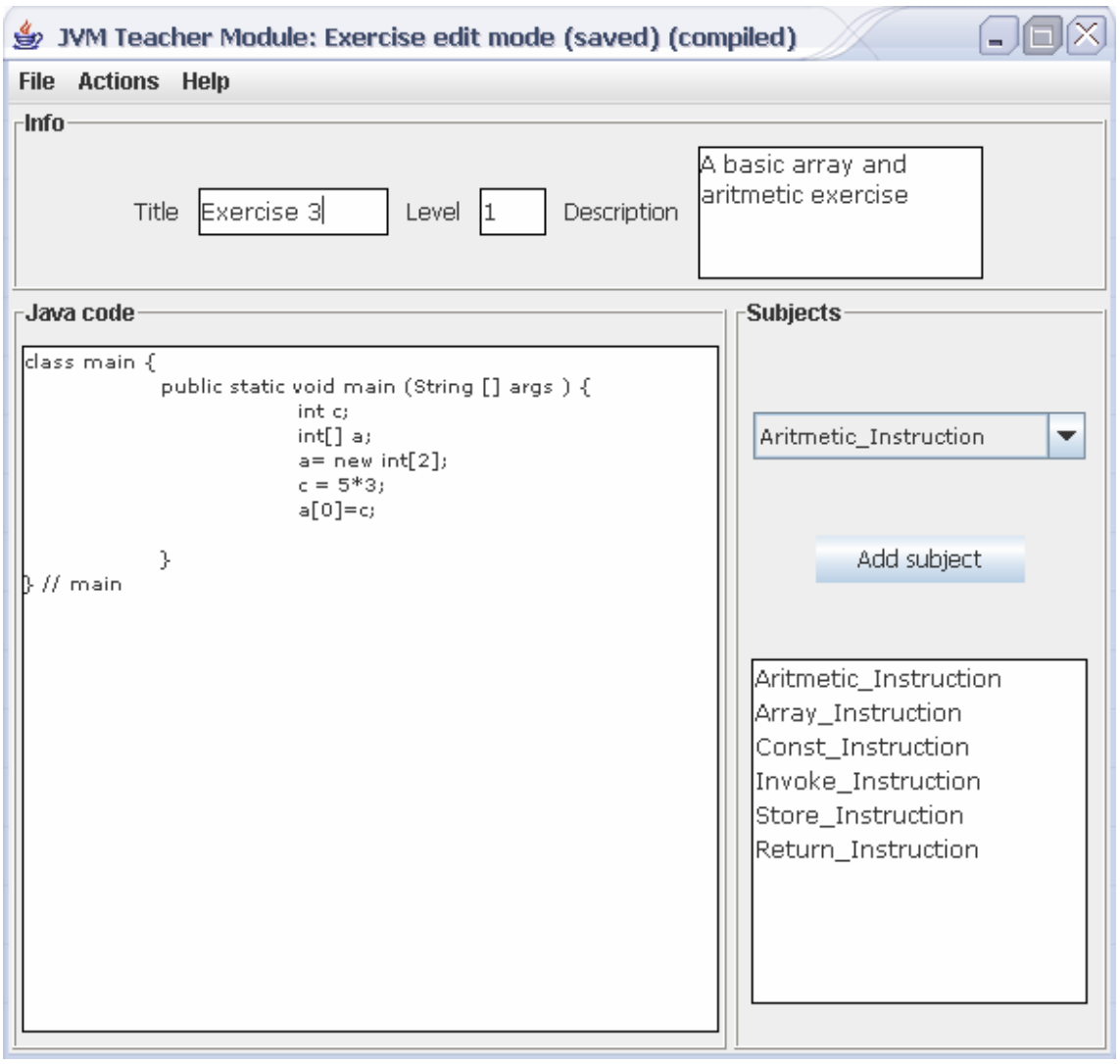


Figura 6.5.4: Editor de ejercicios tras cargar un ejercicio o crearlo

4) Es necesario compilar el ejercicio para comprobar que el código java es correcto. Para ello se pulsa el botón correspondiente.

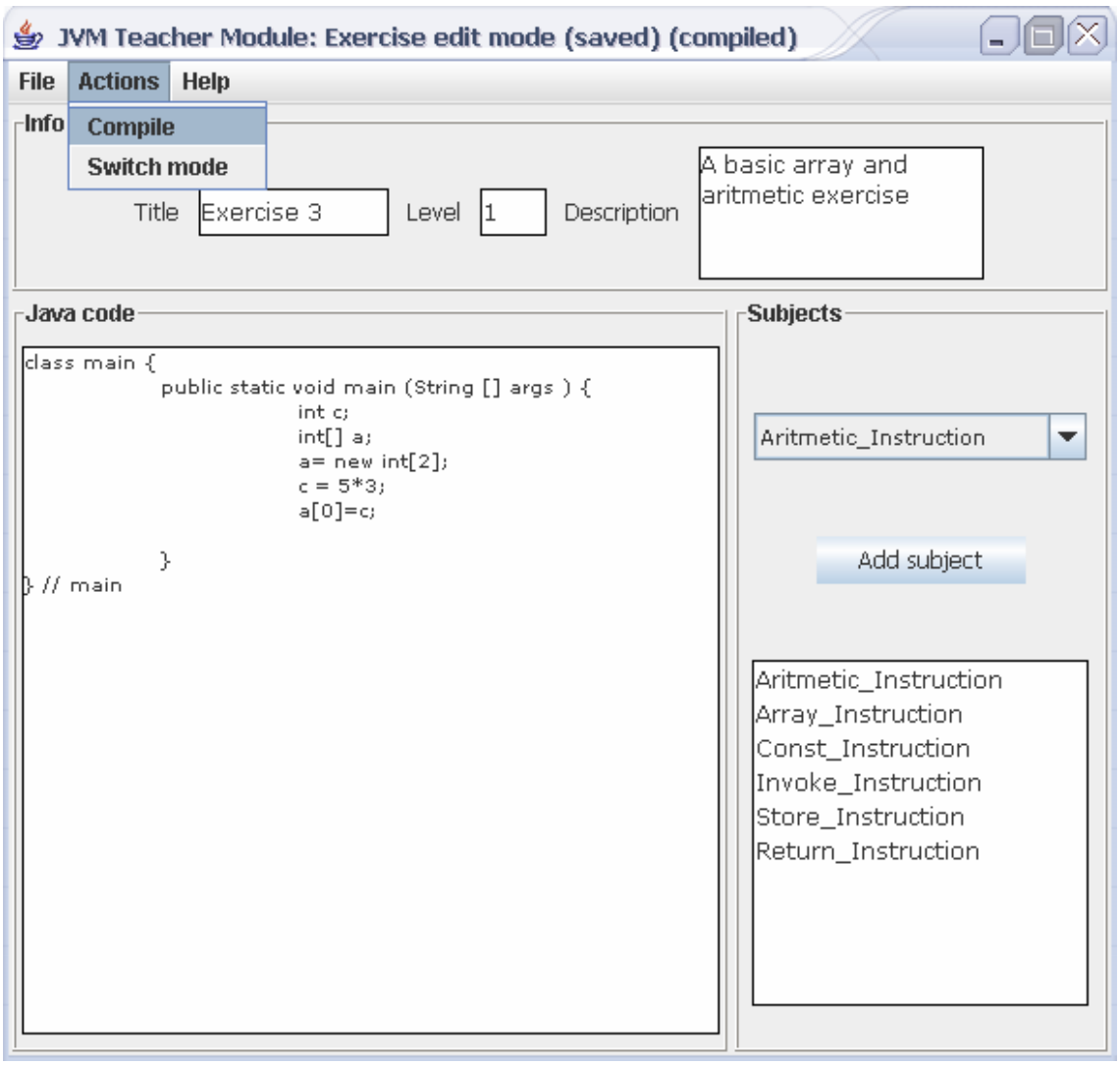


Figura 6.5.5: Editor de ejercicios compilando el código java.

5) En el caso que el código java sea correcto se muestra una ventana indicando que se ha compilado satisfactoriamente, en caso contrario se muestra el error y la línea donde se ha cometido.

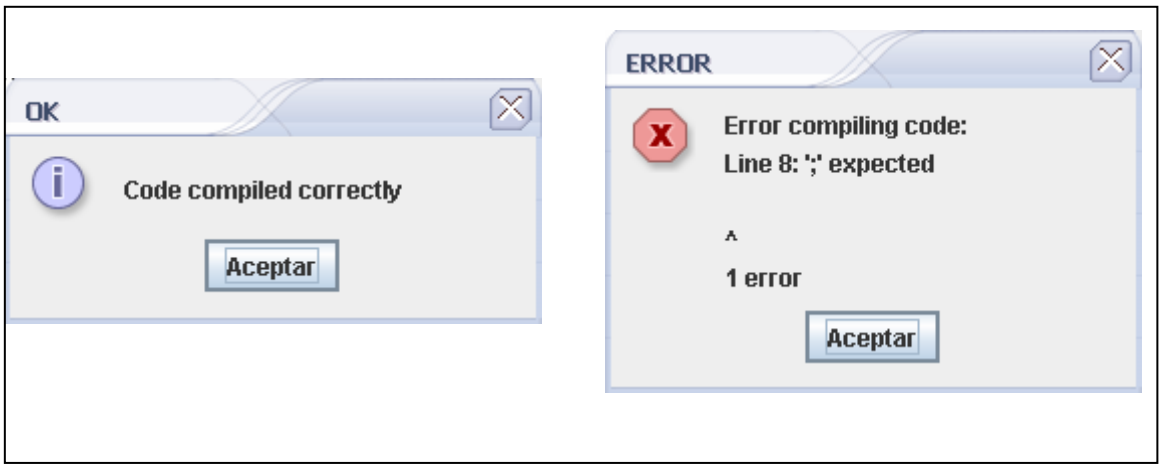


Figura 6.5.6: Mensajes al compilar un ejercicio.

6) Si en la figura 6.5.5 se hubiera pulsado el botón de cambiar de modo “Switch mode”. Se cambia al modo editor de casos donde además del ejercicio se ofrece la posibilidad de crear un error y una o varias ayudas asociadas a ese error.

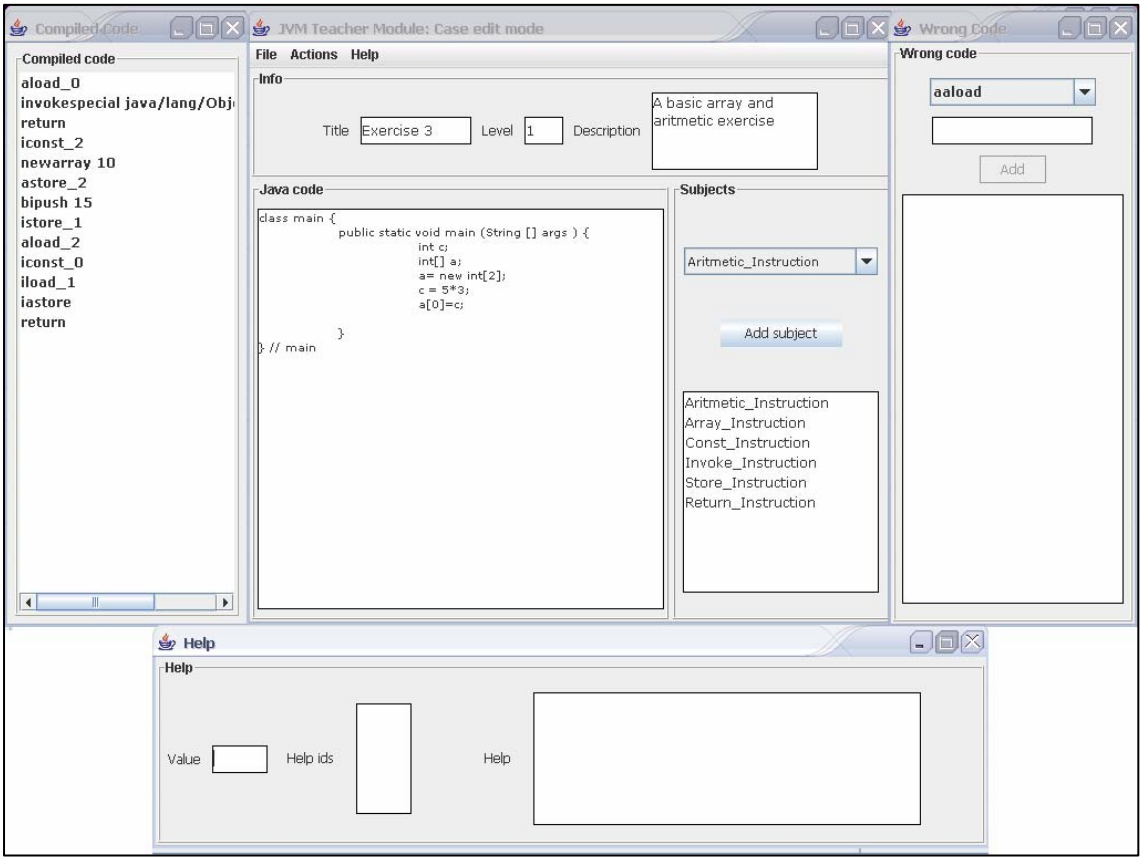


Figura 6.5.7: Editor de casos.

7) El editor de casos muestra a al izquierda el código de la JVM generado tras la compilación del código Java del ejercicio , para facilitar al profesor la creación del error pudiendo copiar estas instrucciones de la JVM y pegarlas en el panel del código erróneo para más tarde modificarlas y crear un error.

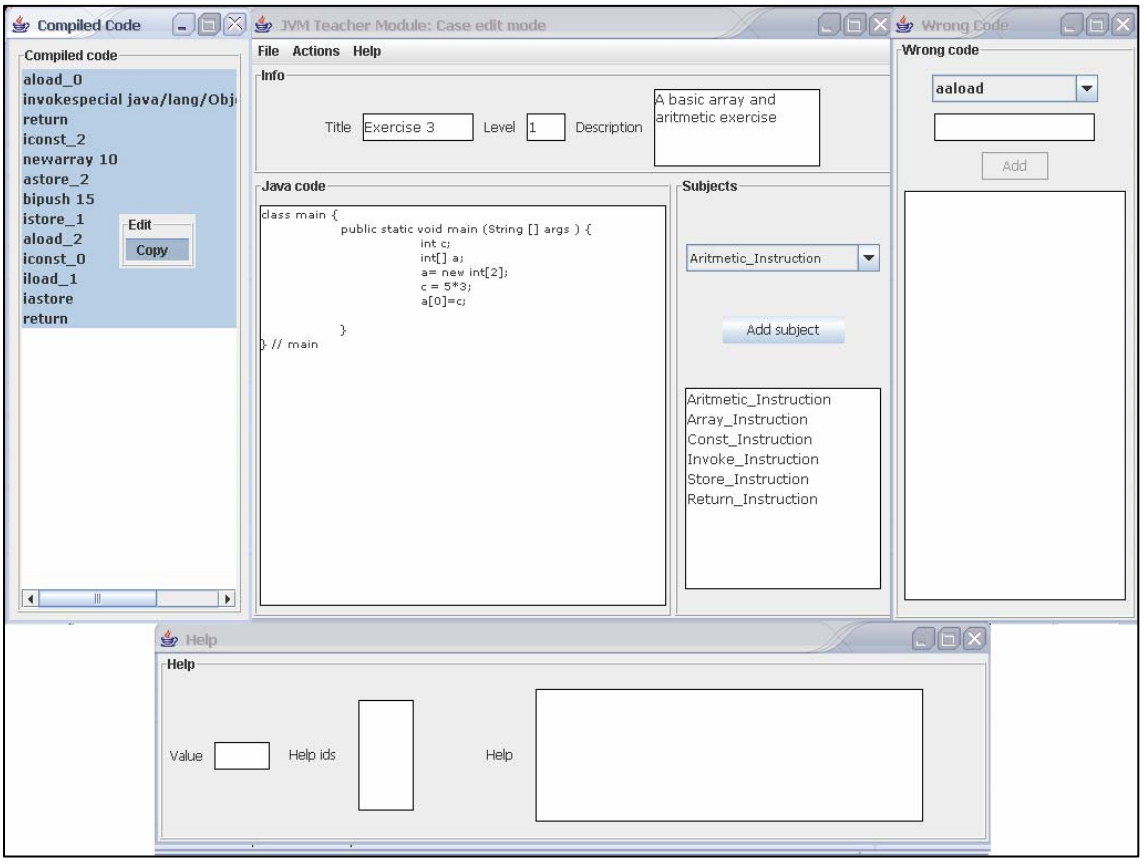


Figura 6.5.8: Editor de casos. Copia del código compilado.

8) Este es el proceso tras copiar el código para pegarlo en el panel de código erróneo.

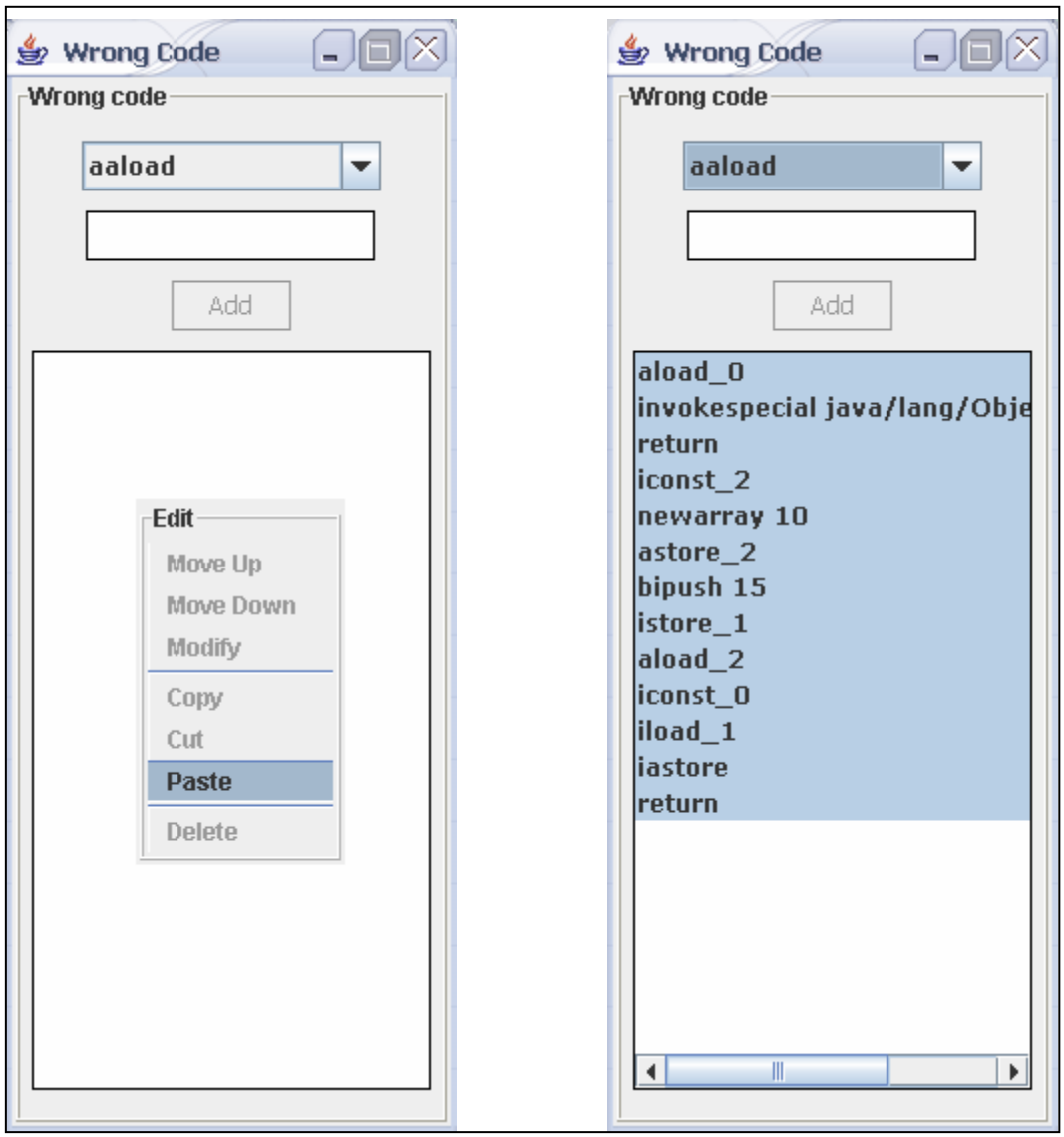


Figura 6.5.9: Pegado del código.

9) Se puede modificar el código anteriormente pegado, para crear un error común.

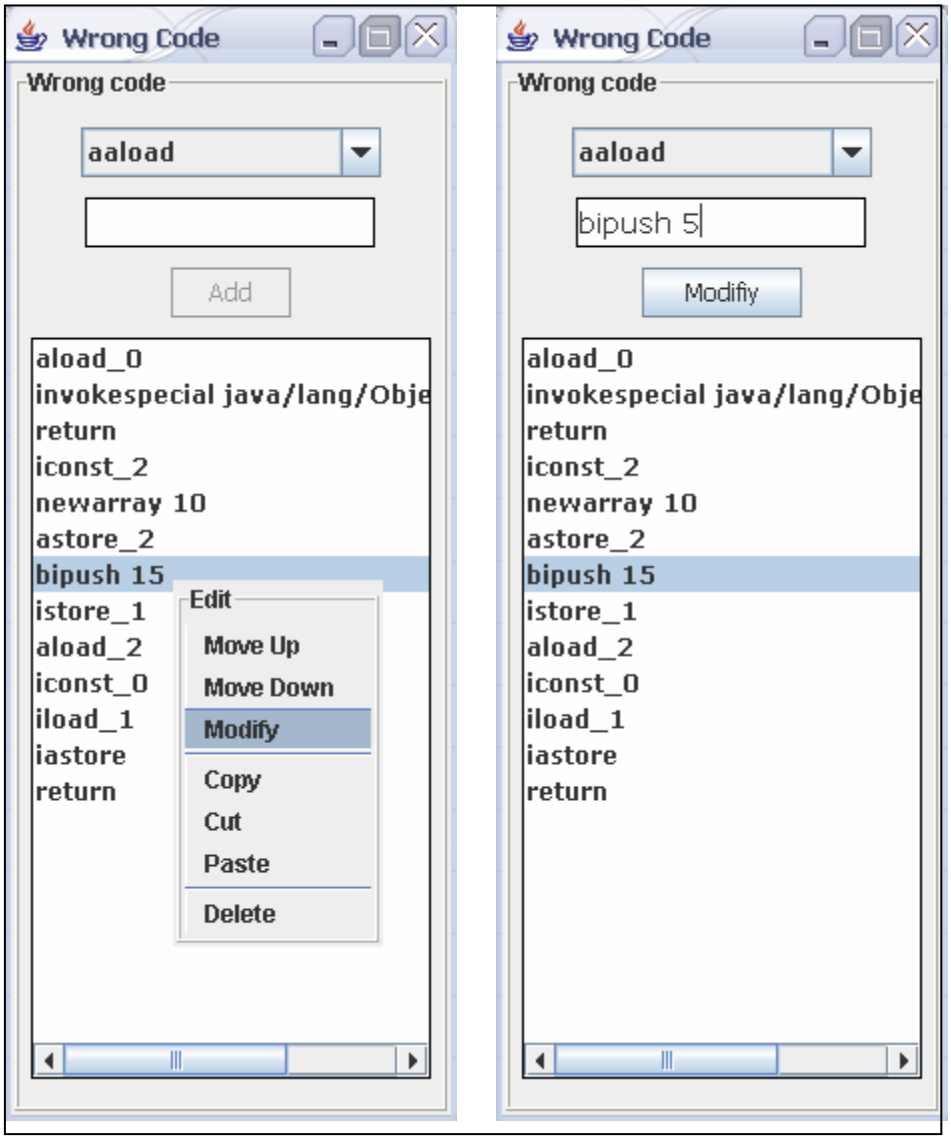


Figura 6.5.10: Modificación del código JVM, para crear un error.

10) Para la creación de las ayudas, se pulsa el botón derecho del ratón sobre el campo “Help ids” y se selecciona o bien “New” que crea una nueva ayuda simplemente escribiéndola en el campo “Help” (**Figura 6.5.11**) o se puede pulsar “Add” para seleccionar una ayuda existente en la base de casos. (**Figura 6.5.12**)

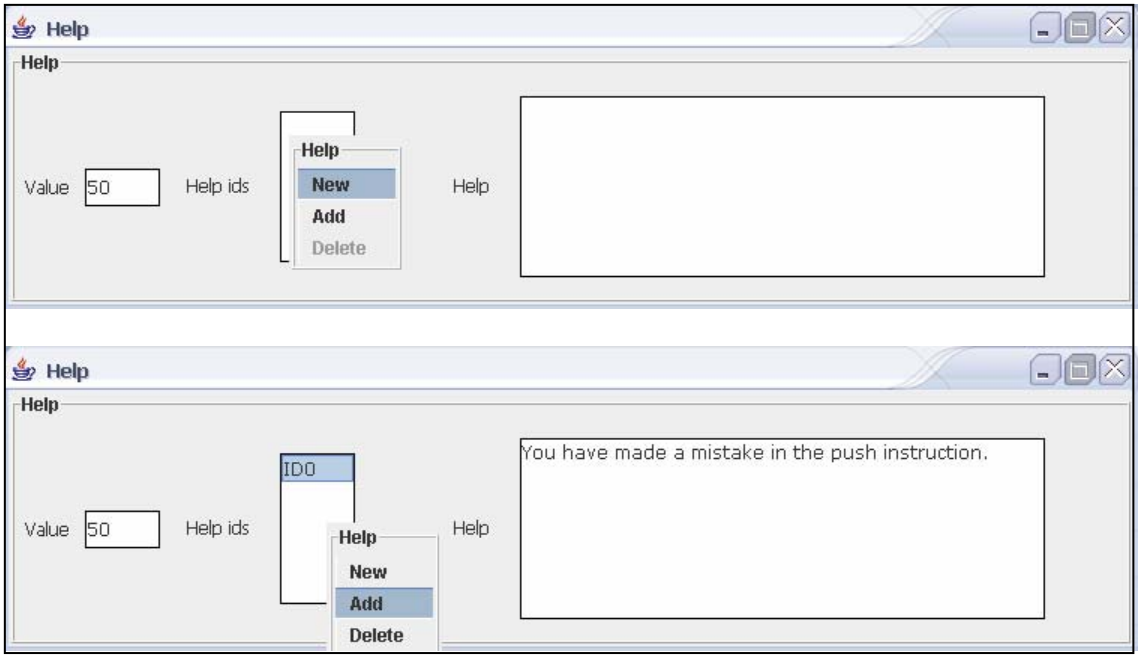


Figura 6.5.11: Creación de una nueva ayuda.

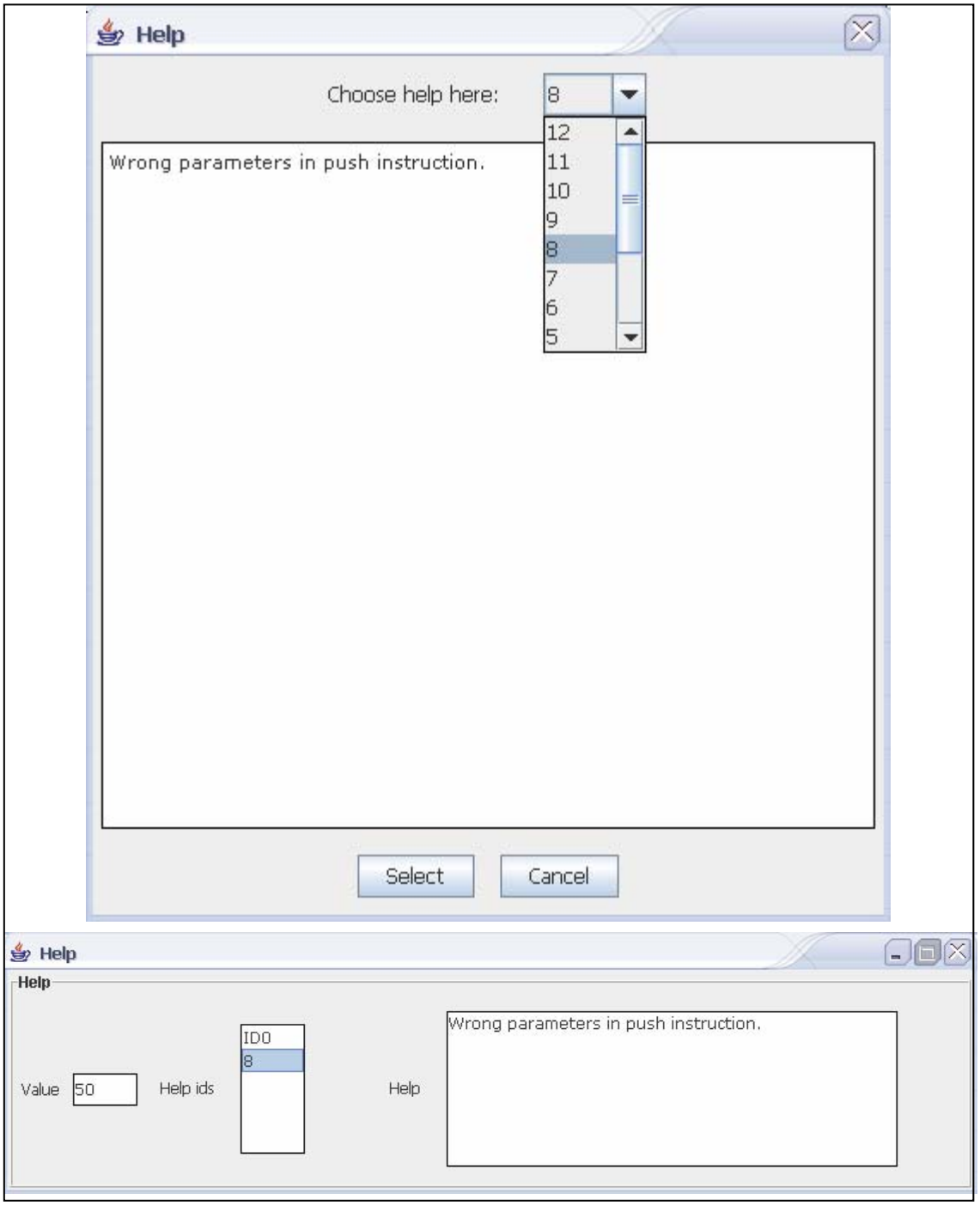


Figura 6.5.12: Carga de una ayuda existente en la base de casos.

11) Cuando el ejercicio y el caso están creados se graba en la base de casos en memoria, la cual al cerrar el editor se grabara en un archivo XML.

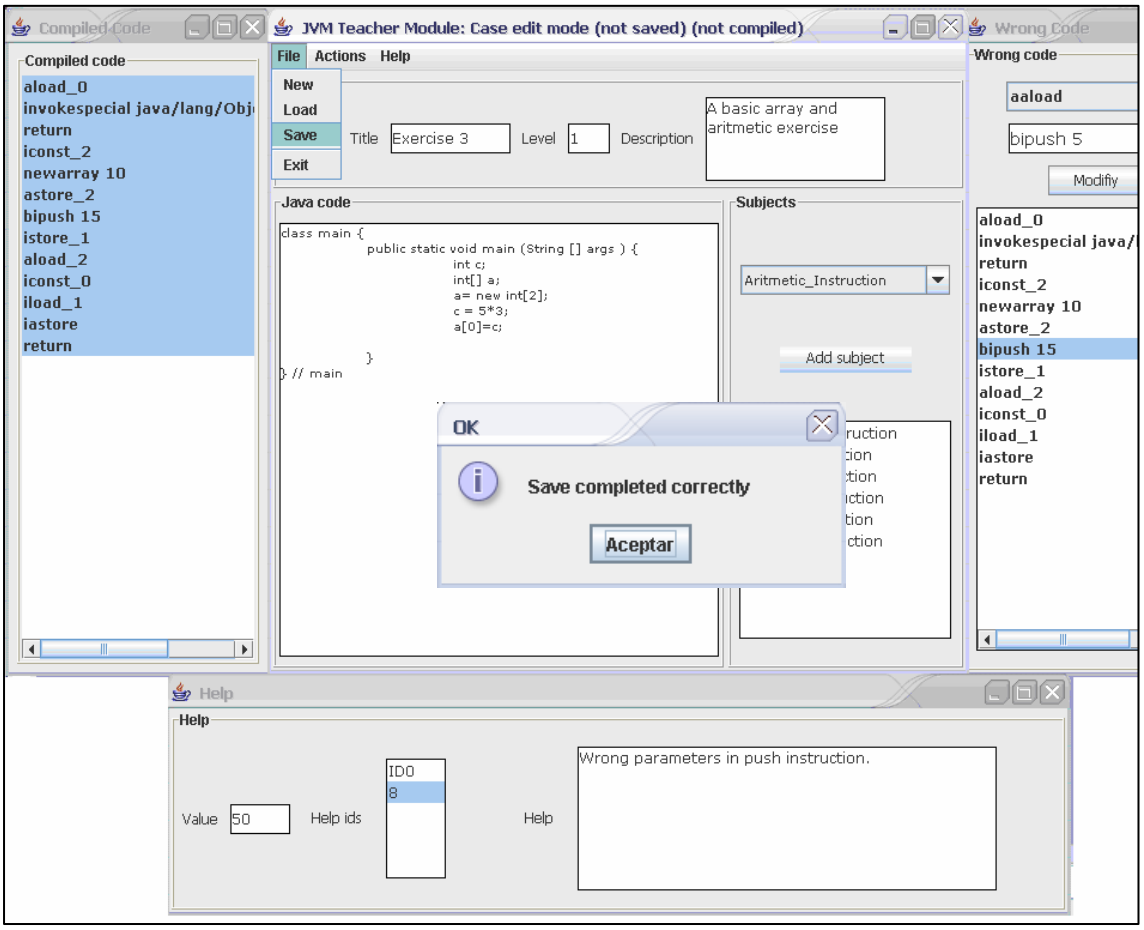


Figura 6.5.13: Grabación de un caso y ejercicio creado.

12) También se puede cargar un caso existente en la base de casos para modificarlo borrarlo, o crear a partir de el nuevos casos de ayuda.

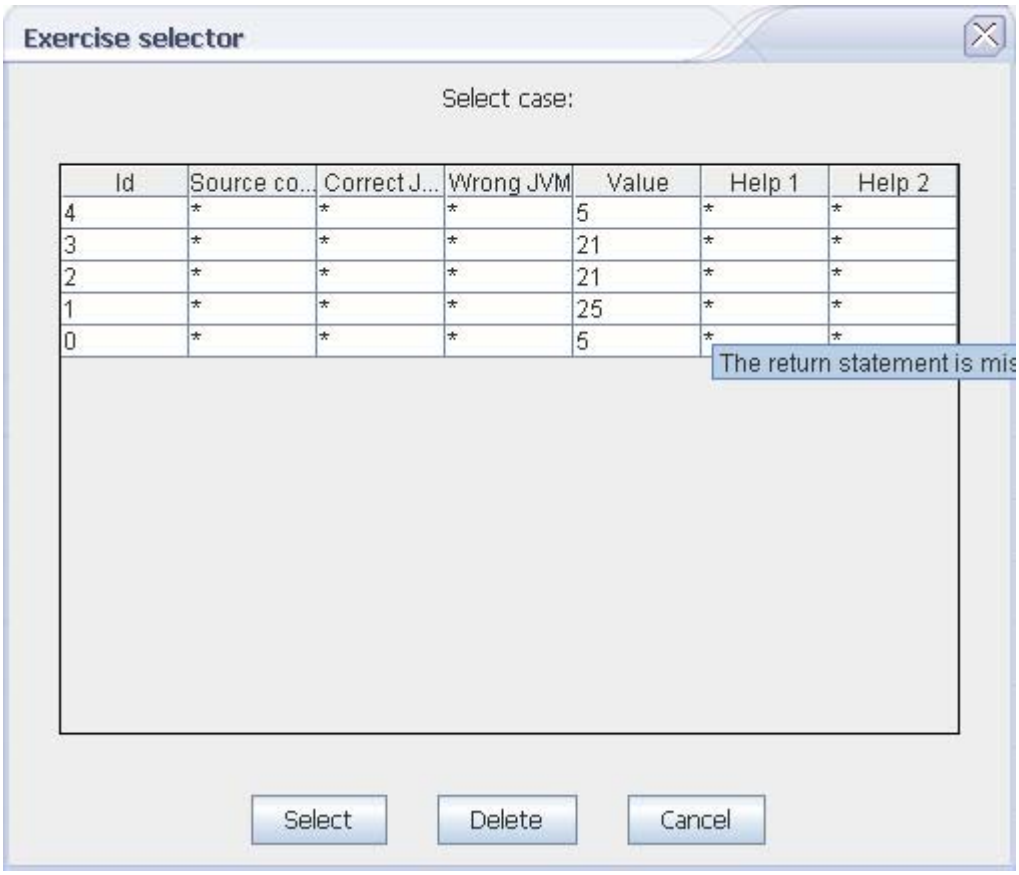


Figura 6.5.14: Selección o borrado de un caso existente en la base de casos.

6.6 MÓDULO DE APRENDIZAJE

6.6.1 Introducción

Este módulo es el encargado de interactuar con el alumno, para ello hay que definir un interfaz gráfico adecuado que cumpla con los requisitos especificados **(Capítulo 2.3.5).** A continuación detallamos sus características y los detalles de su implementación.

6.6.2 Características

El interfaz grafico del módulo de aprendizaje contiene las componentes visuales necesarias para facilitar al alumno las dos tareas principales relacionadas con este módulo: resolver ejercicios creados previamente por el profesor y obtener la ayuda necesaria para la resolución de los mismos.

Con respecto a los ejercicios hemos dispuesto dos formas de obtener un ejercicio, eligiéndolo de una lista o dejando que el sistema proponga un ejercicio para un perfil de usuario dado. Para representar el código JVM que propone el alumno como solución al problema hemos utilizado una lista en la que hemos incluido un menú para editar las instrucciones con multitud de opciones, en este interfaz también hemos colocado una lista desplegable con todas las instrucciones cargadas de la ontología. **(Capítulo 4.3.2)**

En el desarrollo de este módulo hemos primado ante todo la facilidad de uso, por eso hemos incluido ayudas para todos los menús y distintas formas de acceso a las mismas para facilitar la utilización de la aplicación.

6.6.3 Implementación

La implementación de este módulo se ha llevado a acabo con las herramientas de generación automática de código para interfaces graficas del JBuilder 9.0, siempre siguiendo el diseño detallado en **(Capítulo 5.3),** y al igual que en el módulo del profesor separando claramente las componentes graficas de sus funcionalidades asociadas, usando el patrón modelo vista controlador detallado en el capitulo. **(Capítulo 7.2.1.2)**

6.6.4 Validación y Verificación. Pruebas

Las pruebas de este módulo se ha realizado en dos fases:

- Primero por separado para probar que se ajustaba a las especificaciones y los diseños a nivel de funcionalidades y facilidad de uso.
- Y finalmente en conjunto con el resto de módulos para probar las conexiones entre ellos, como las necesarias para aportar los distintos tipos de ayudas, la carga de ejercicios

6.6.5 Ejemplos de funcionamiento del Sistema

A continuación mostramos los ejemplos de funcionamiento más característicos del sistema. A partir del ejercicio creado en el ejemplo de ejecución del editor de ejercicios y casos.

1) En primer lugar el alumno tiene que crear o cargar su perfil de usuario donde se almacenará el nivel que tiene y los ejercicios y temas que domina.

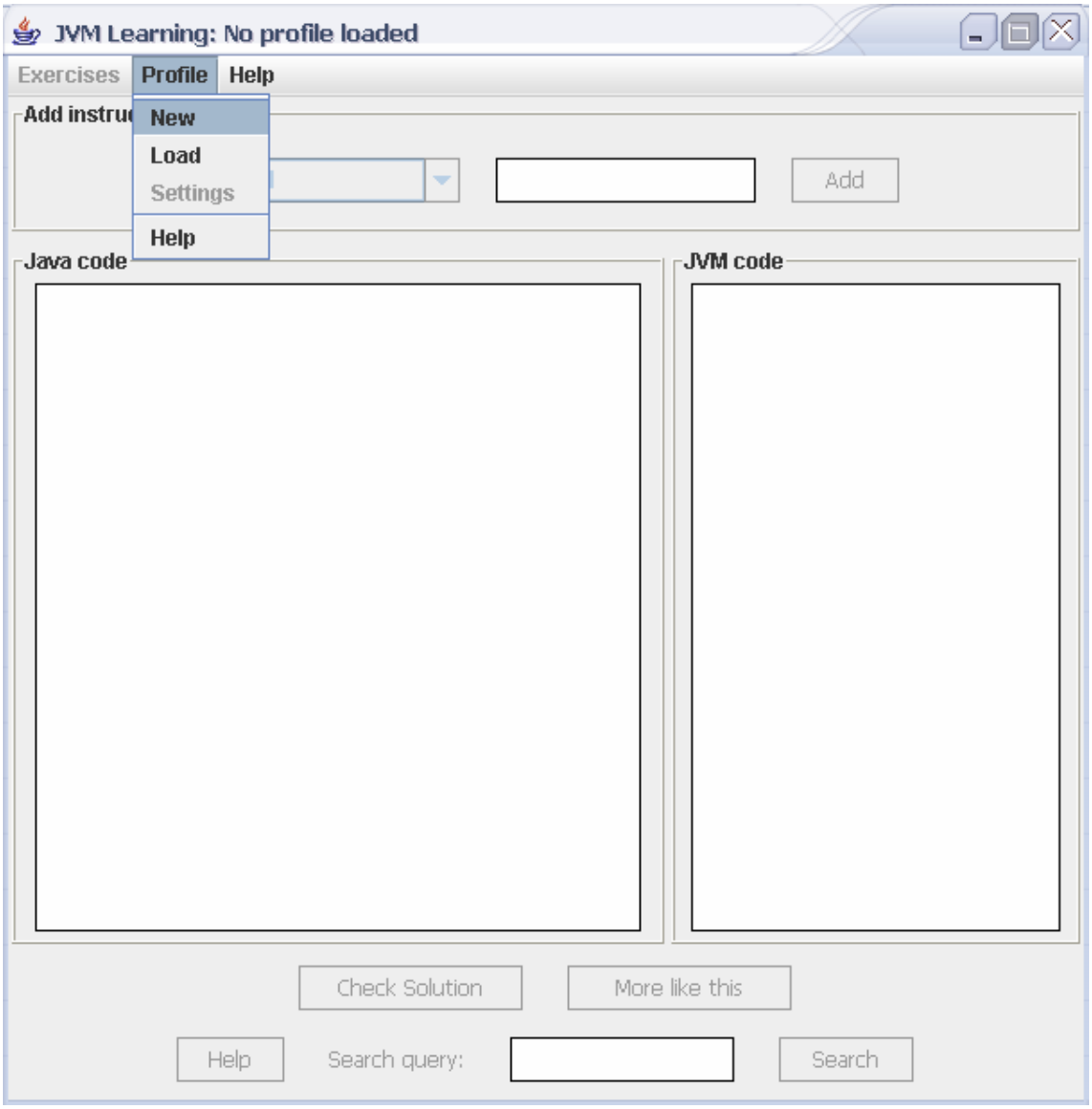


Figura 6.6.1: Creación o carga del perfil de usuario.

Al crear un perfil aparece una ventana donde se introduce el identificador de usuario y pulsando la tecla “F6” puede elegir los temas que domina y el nivel que tiene. (**Figura 6.6.2**), si todo es correcto y uno es un usuario repetido un panel nos muestra que todo es correcto (**Figura 6.6.3**)

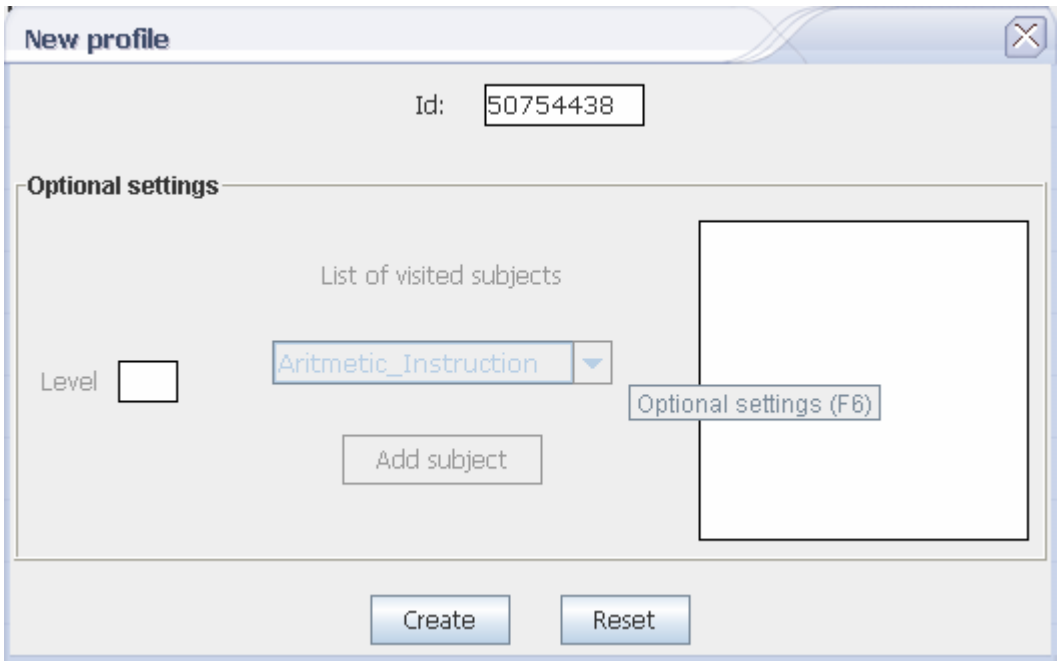


Figura 6.6.2: Creación del perfil de usuario.

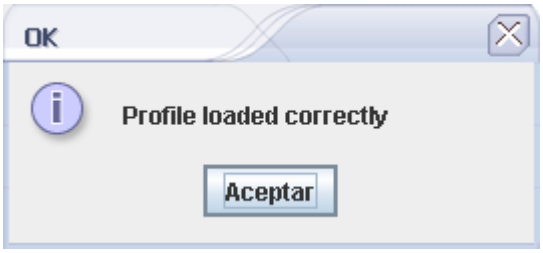


Figura 6.6.3: Panel que muestra que el usuario se ha cargado correctamente.

2) A continuación se puede elegir o pedir que nos propongan un ejercicio según nuestro nivel (gracias al perfil de usuario).(**Figura 6.6.4 y 6.6.5**)

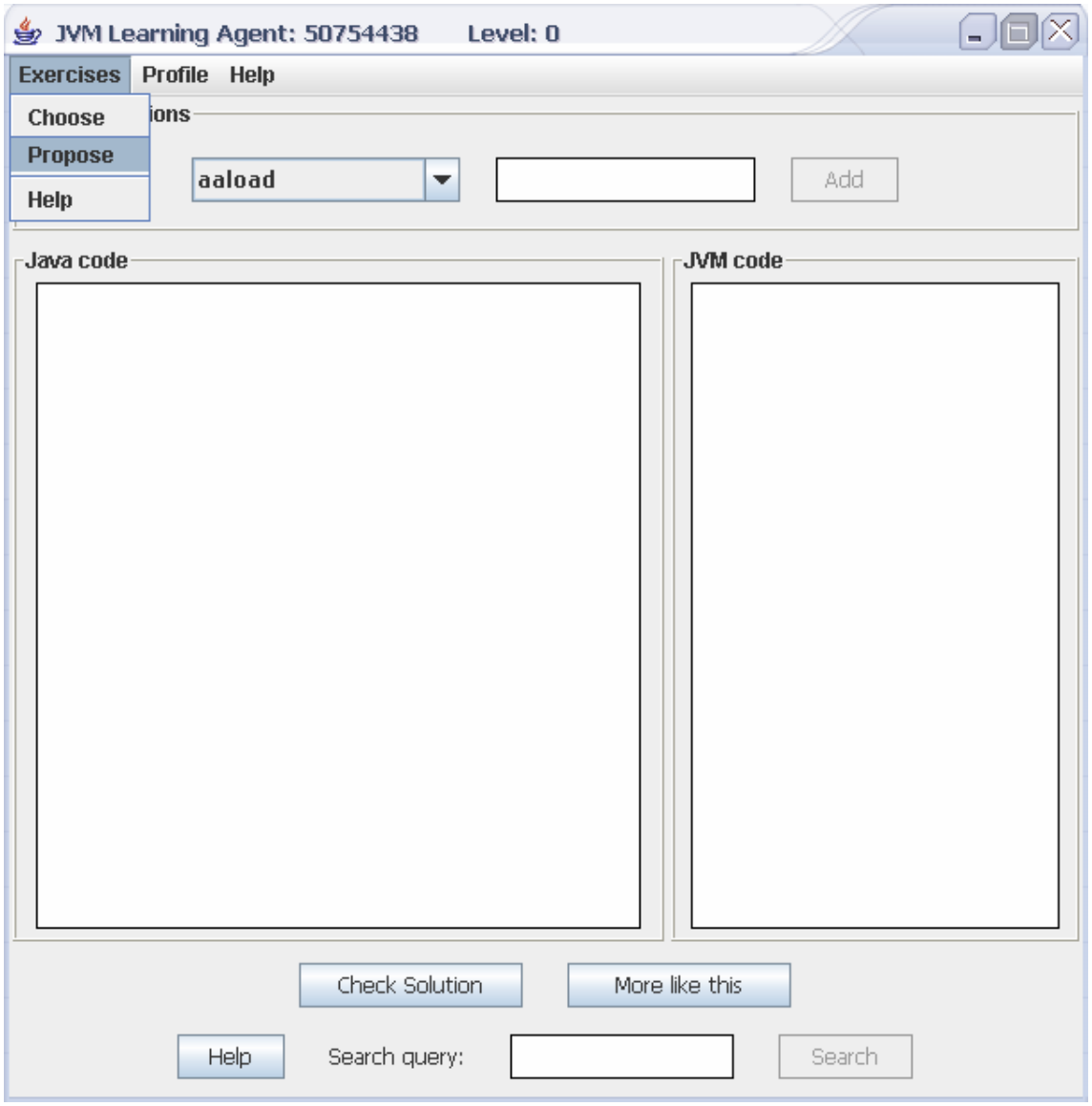


Figura 6.6.4: Selección de ejercicio o propuesta

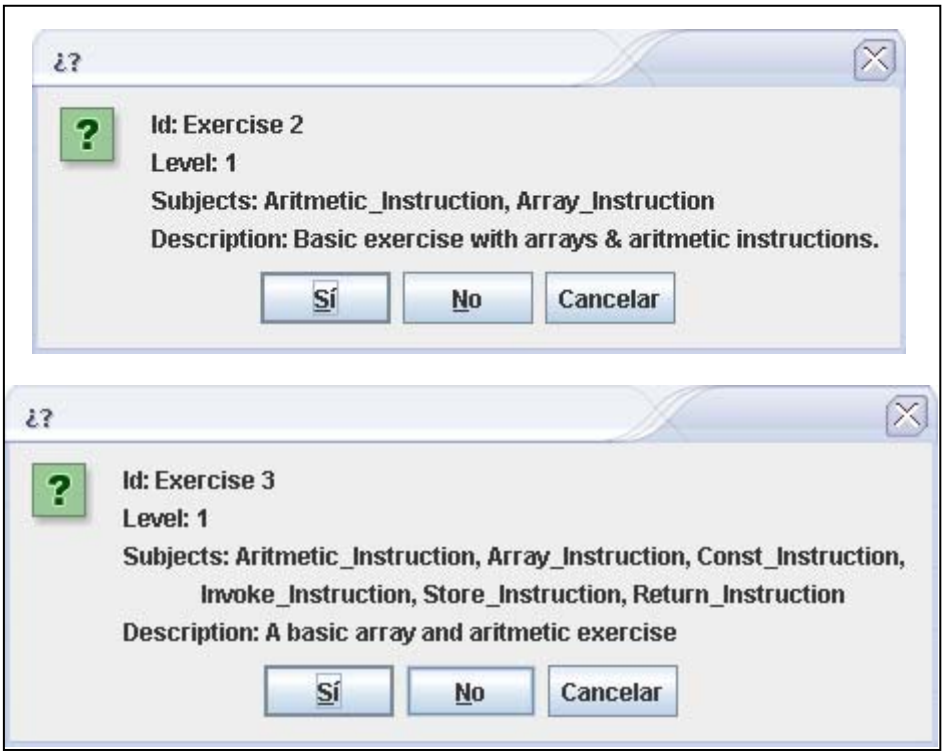


Figura 6.6.5: Ejercicios propuestos por el sistema

3) A continuación el ejercicio seleccionado o propuesto se carga mostrándose en el módulo de aprendizaje. (**Figura 6.6.6**)

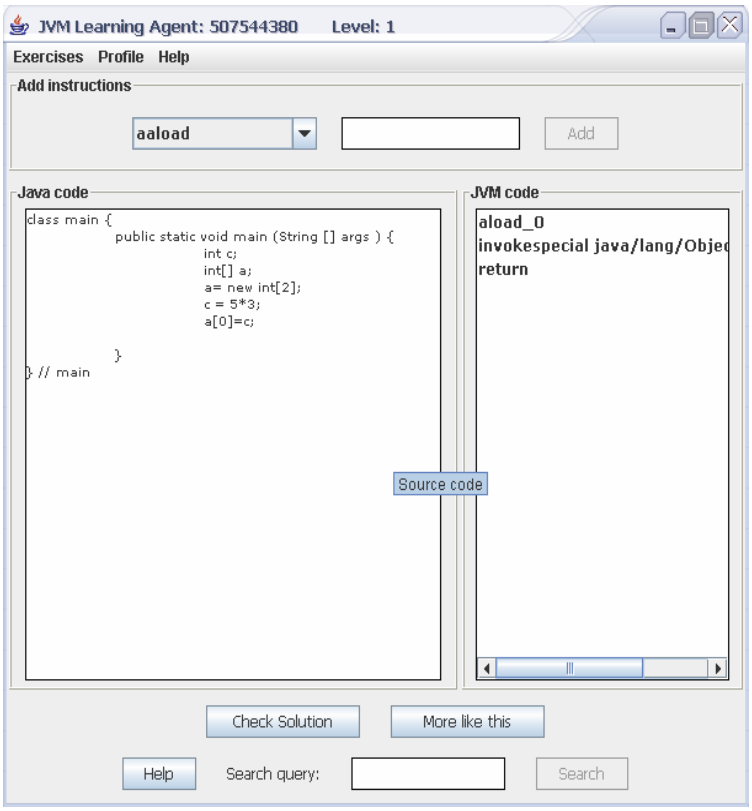


Figura 6.6.6: Ejercicio cargado

4) El alumno se dedica entonces a la resolución del ejercicio seleccionando las instrucciones que él crea conveniente de la lista deplegable o escribiéndolas el mismo.(**Figura 6.6.7**)

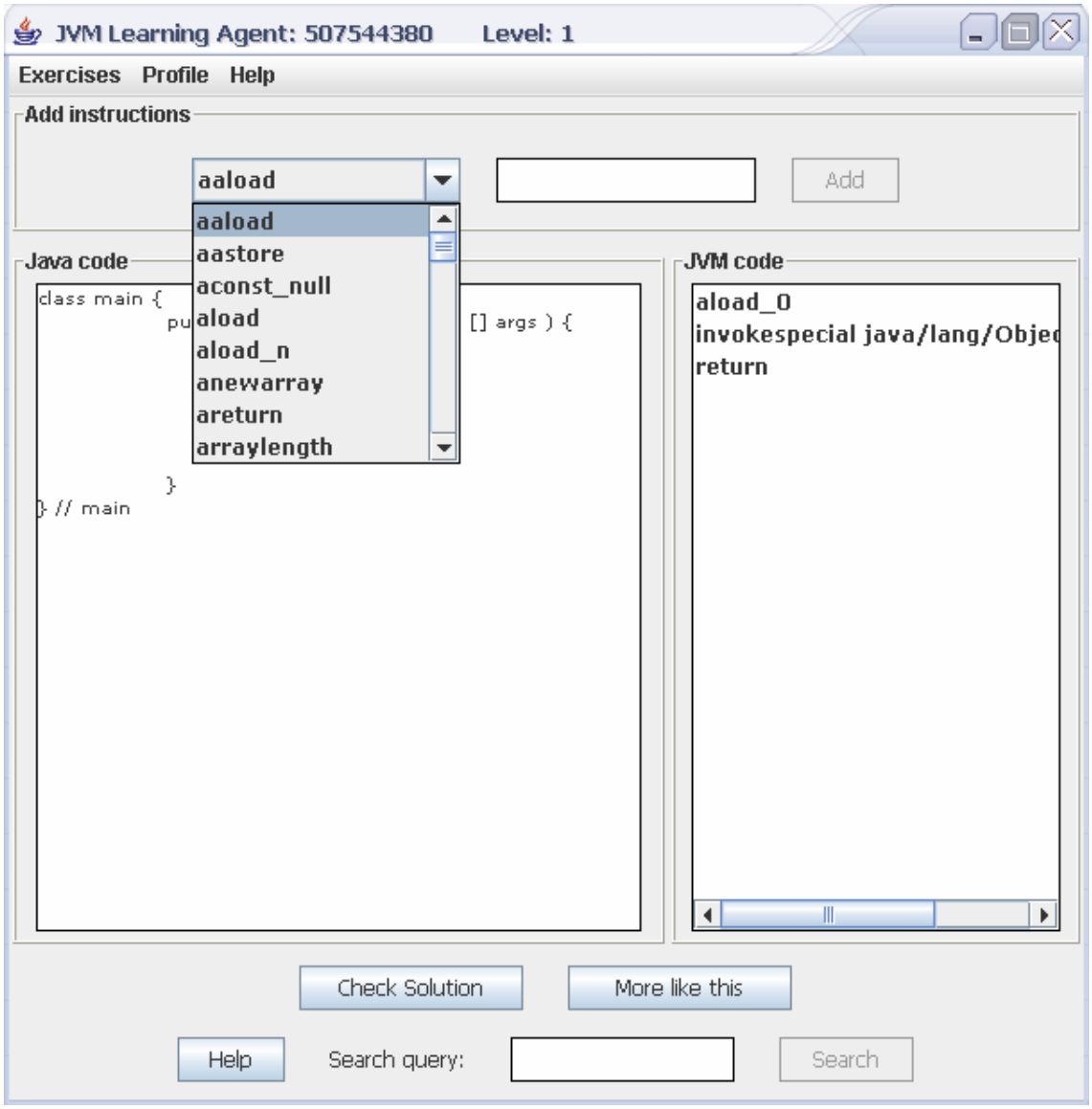


Figura 6.6.7: Selección de instrucciones.

5) En cualquier momento el alumno puede solicitar ayuda pulsando el botón “Check Solution”.

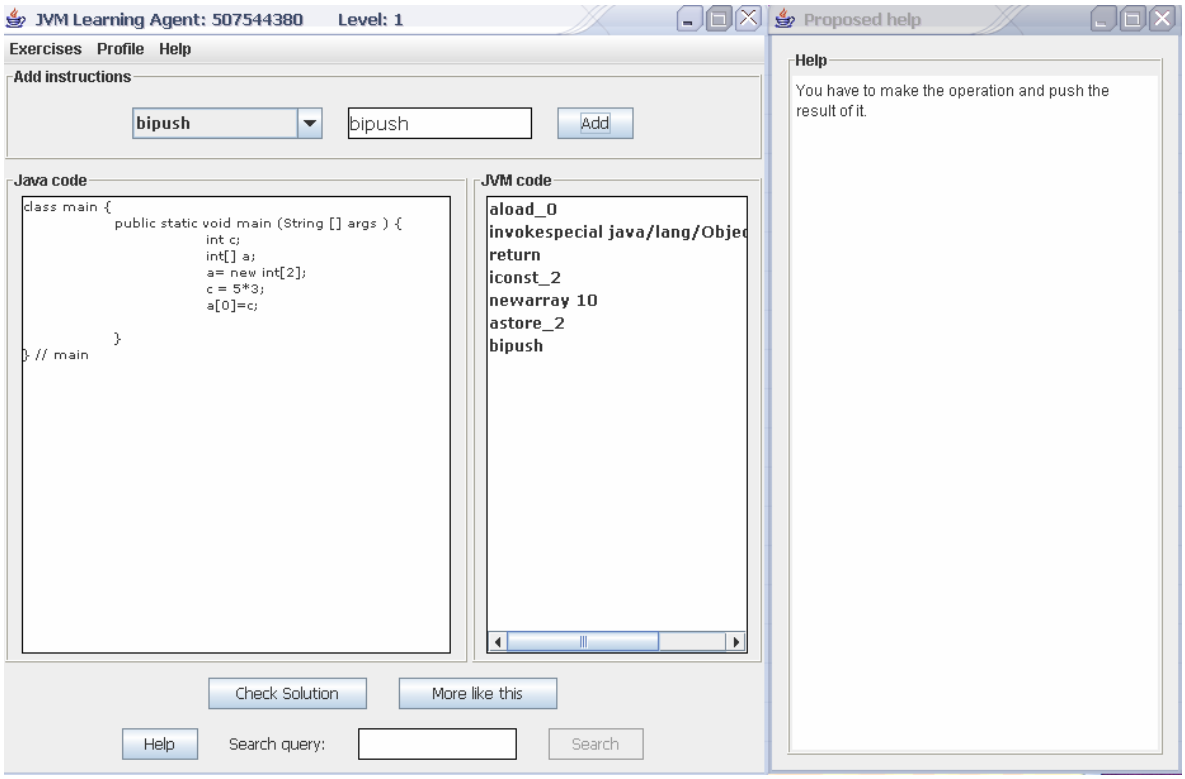


Figura 6.6.8: Solicitud y muestra de ayuda.

6) Se puede solicitar más ayuda similar con el botón “More Like This”(.**Figura 6.6.9)**

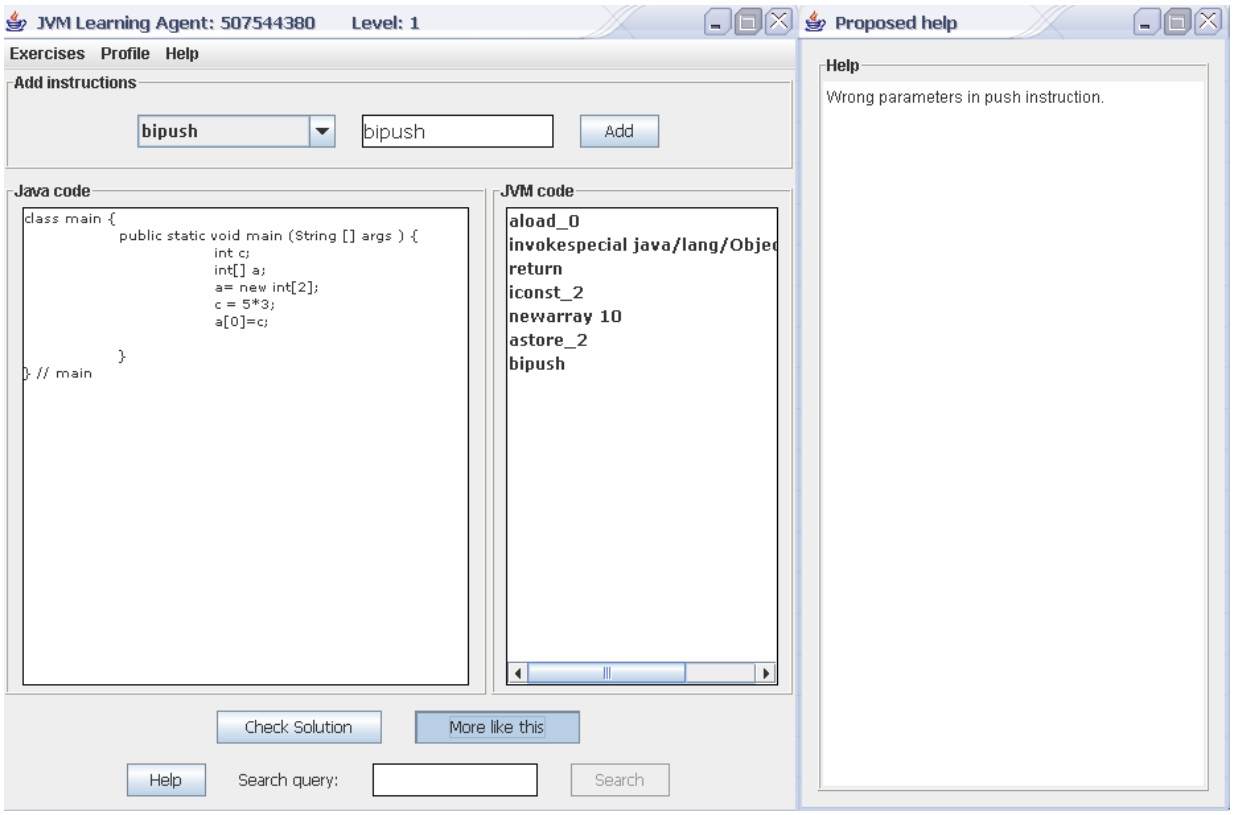


Figura 6.6.9: Solicitud de más ayuda similar a la anterior con el botón “More Like This”.

7) La ayuda que se solicita puede ser también una pregunta concreta sobre una instrucción o una descripción de la instrucción que estamos buscando. Por ejemplo ya que el error del ejemplo nos dice que tenemos mal la instrucción bipush se puede preguntar: “What is bipush for?” en el cuadro llamado “Search query” , después pulsamos el botón “Search” que en la imagen 6.6.8 esta deshabilitado ya que no hay ninguna pregunta escrita. Se observa en la imagen **6.6.10., 6.6.11 y 6.6.12**

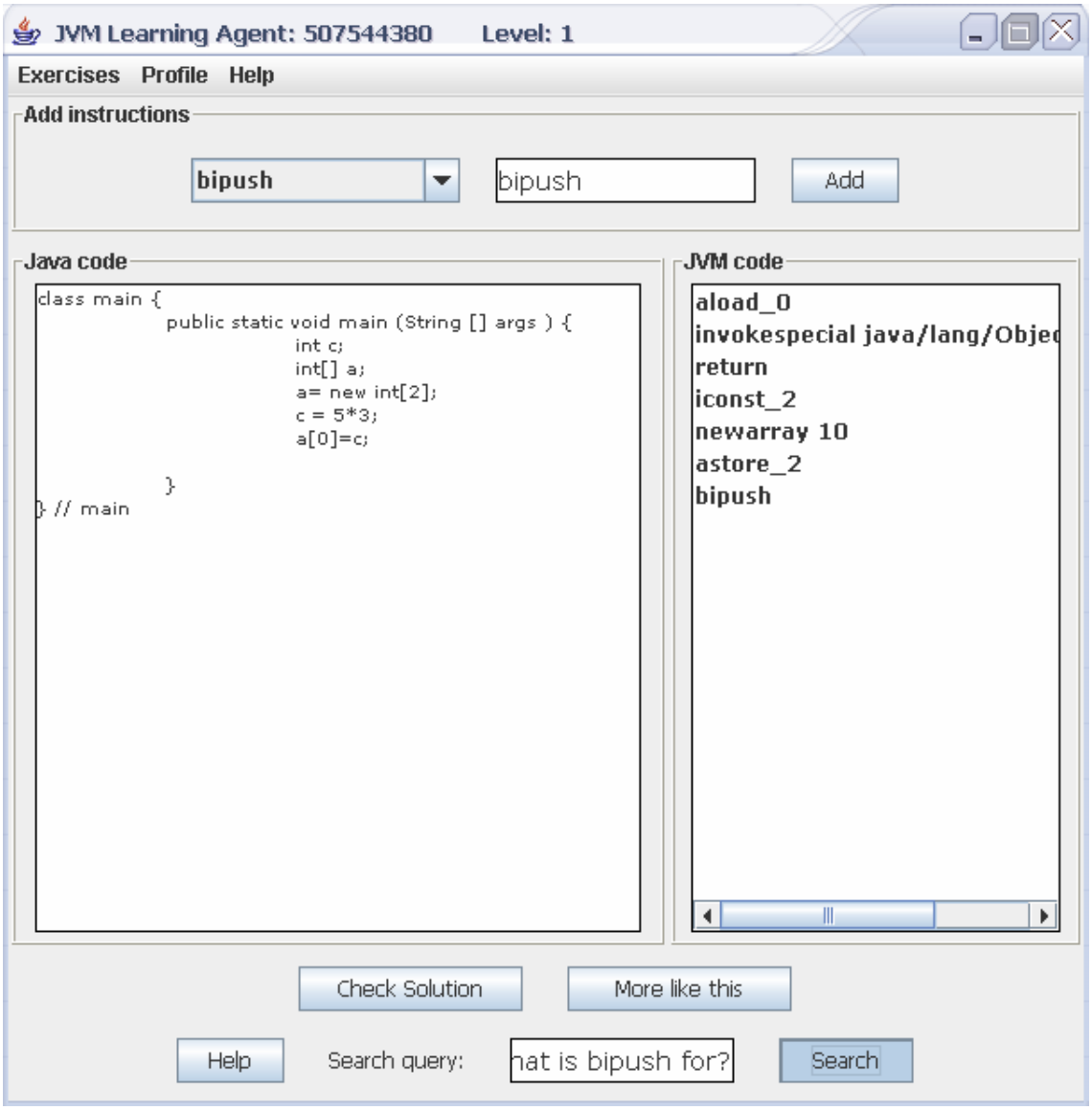


Figura 6.6.10: Solicitud de ayuda textual, mediante una pregunta en ingles.

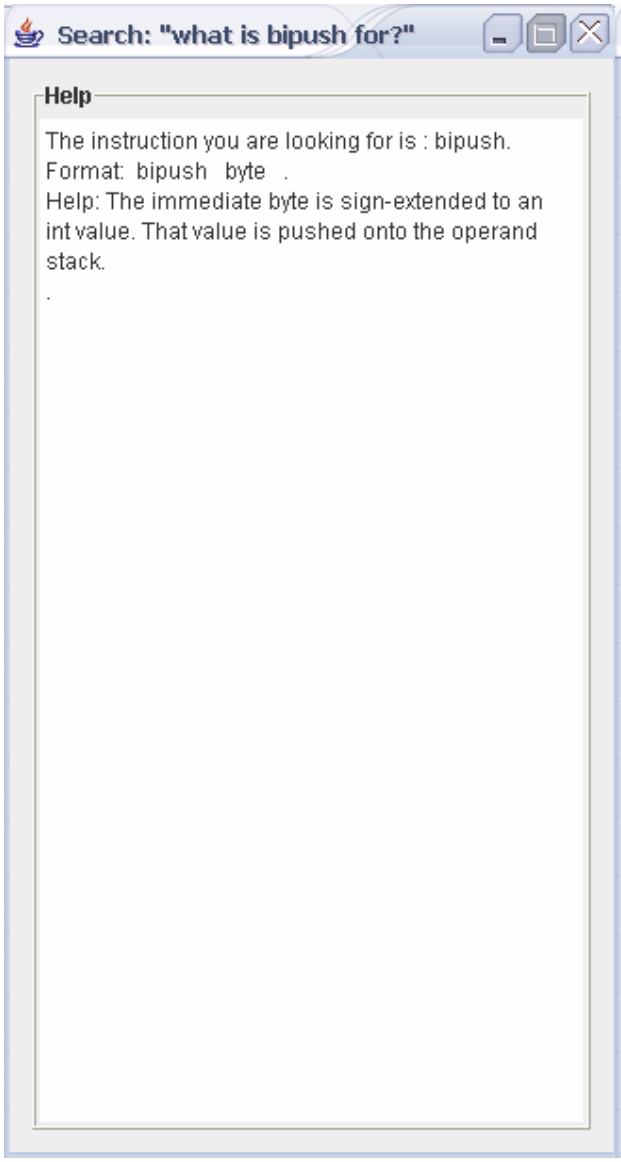


Figura 6.6.11: Respuesta ala pregunta: What is bipush for?

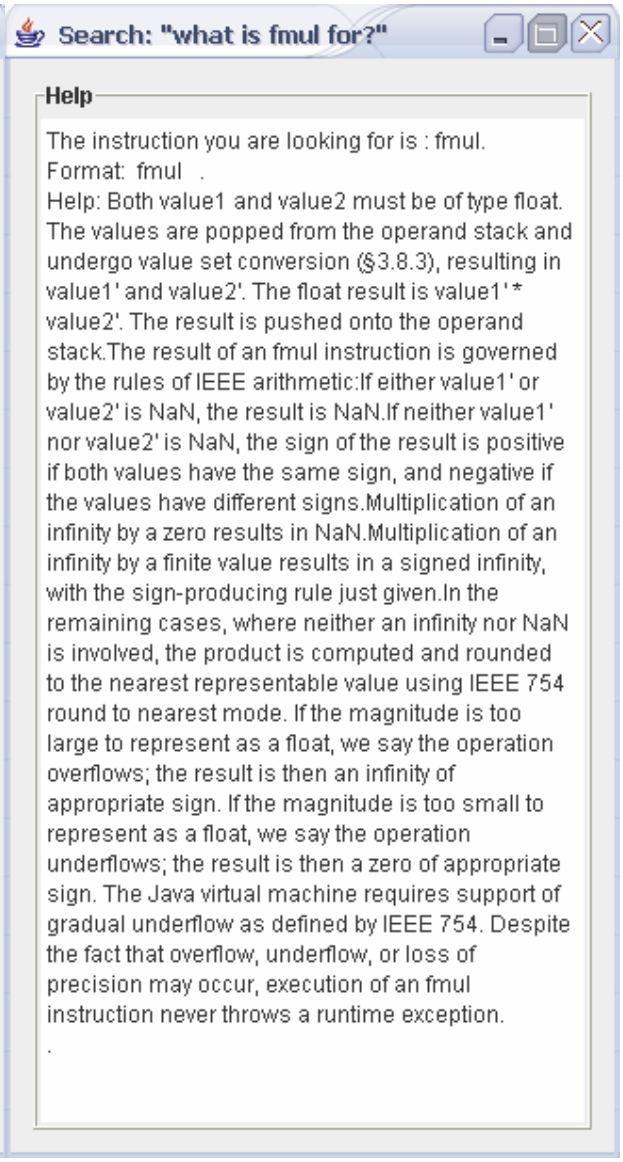


Figura 6.6.12: Respuesta ala pregunta: What is fmult for?

8) El alumno continua resolviendo el ejercicio **Figura 6.6.13** hasta que lo completa correctamente gracias alas ayudas **Figura 6.6.14**

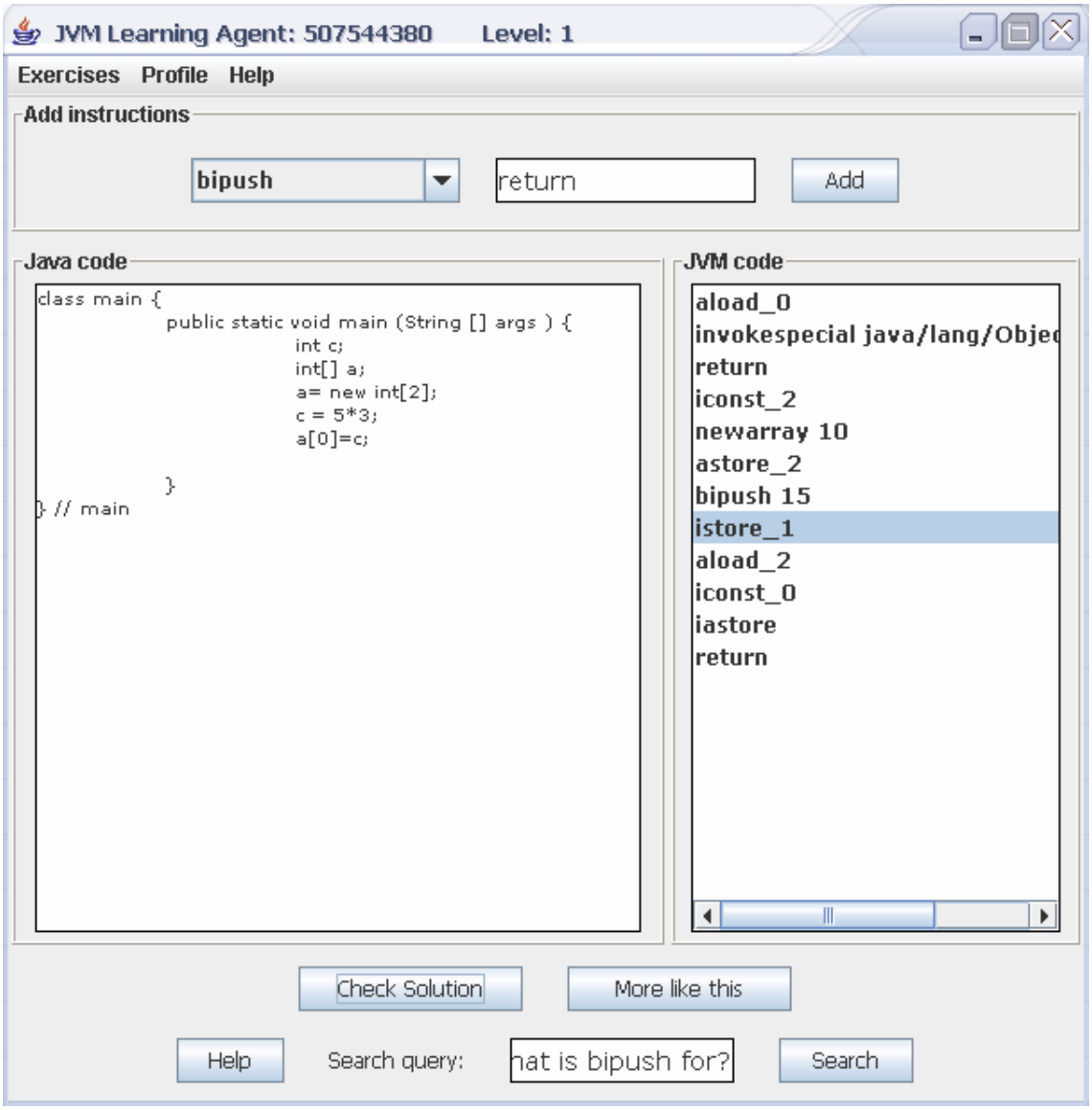


Figura 6.6.13: Módulo de aprendizaje.

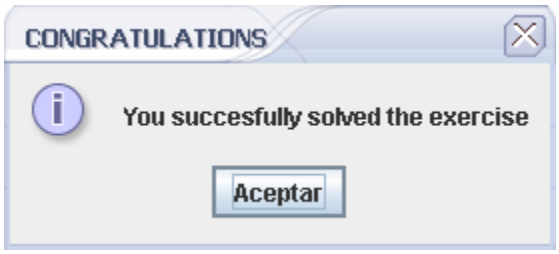
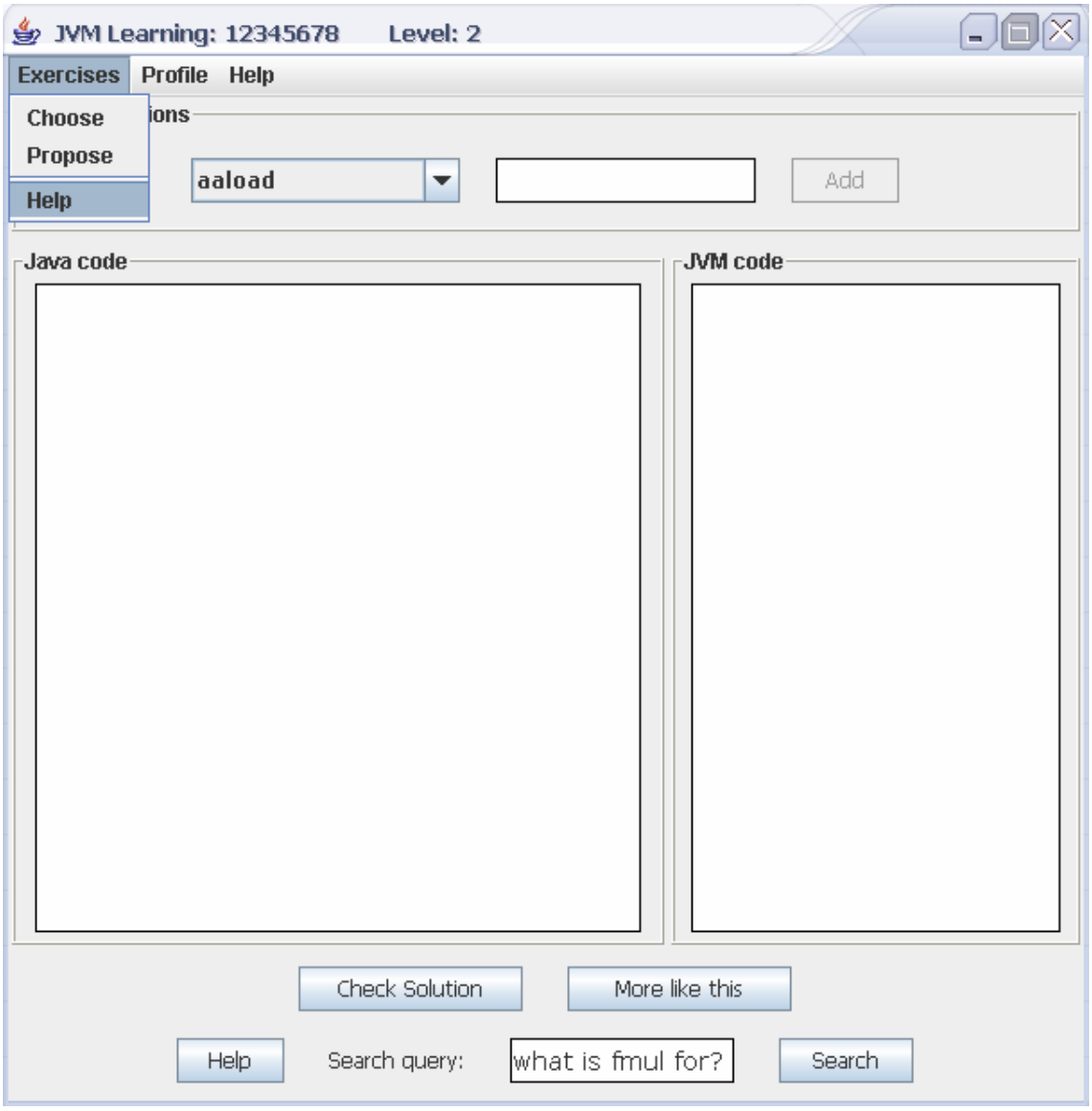
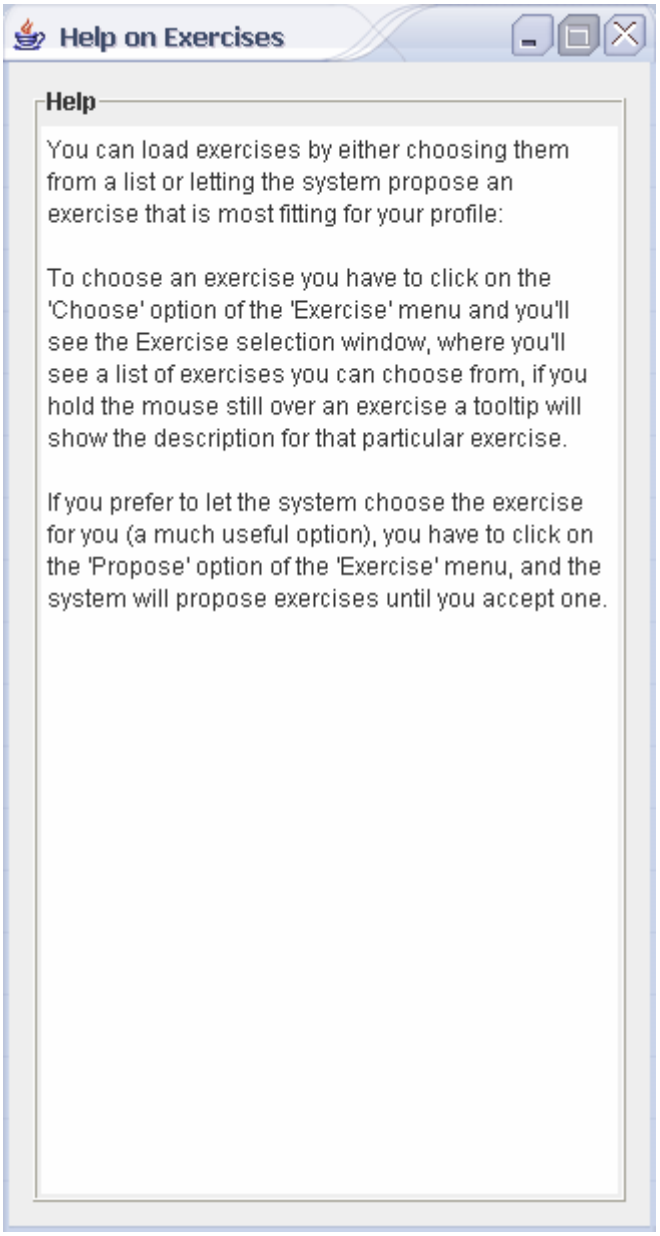


Figura 6.6.14: Ejercicio completado.

Este es el modo de funcionamiento general, además hay algunos casos de uso que no se han mostrado pero si se han explicado en el diseño c**apítulo 2.3.1,** además se puede solicitar ayuda sobre la aplicación **Figuras: 6.6.15 y 6.6.16**

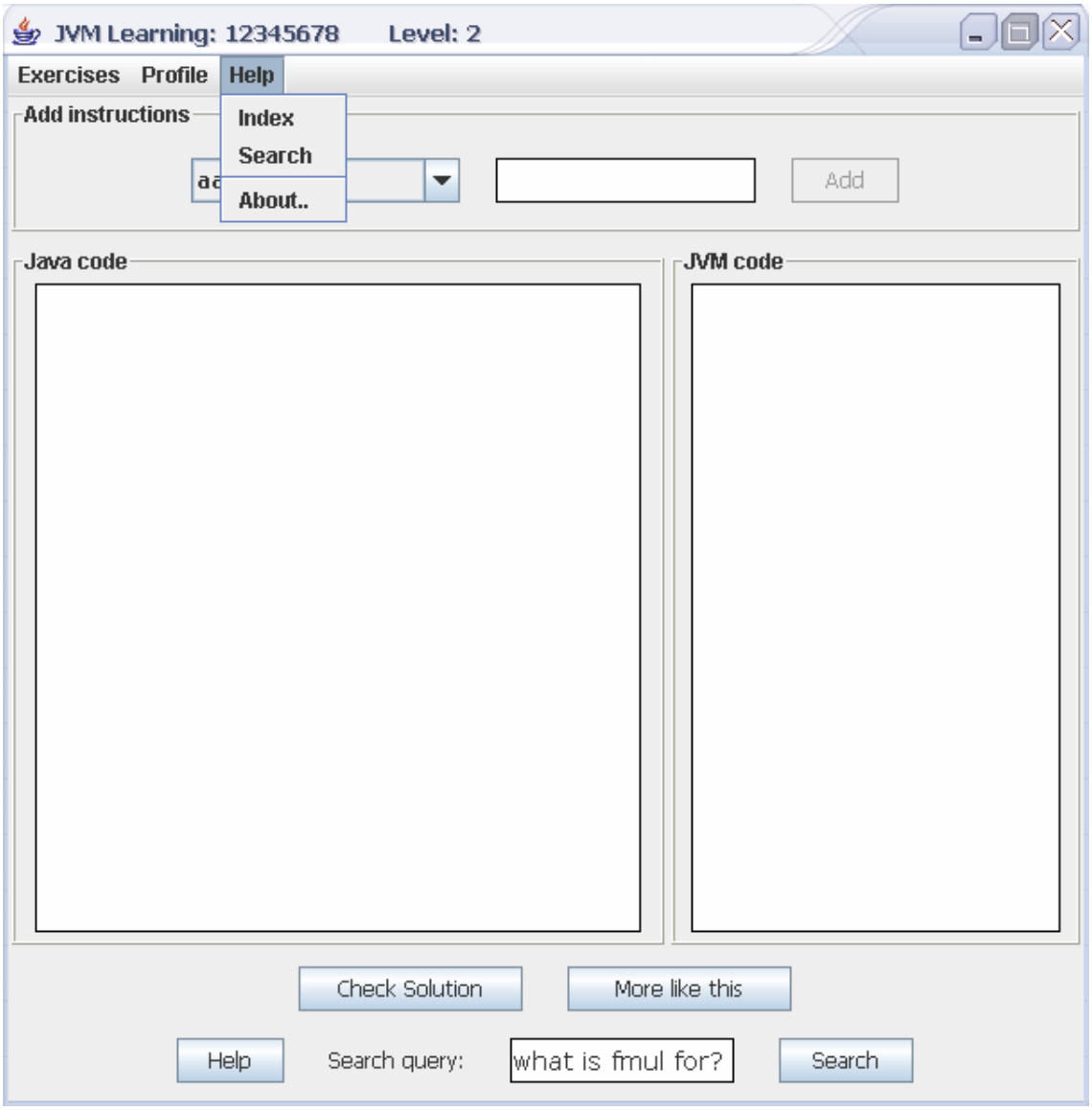


Figuras 6.6.15: Selección de ayuda sobre el módulo de aprendizaje.



Figuras 6.6.16: Ayuda sobre el módulo de aprendizaje.

Además si se pulsa sobre Help, se puede hacer una pregunta sobre una instrucción ver las personas que han desarrollado este módulo y acceder a la documentación tanto de la aplicación (sistema de enseñanza interactivo) como a la documentación sobre la maquina virtual de JAVA**.** Ambas en formato HTML que se abrirá con el explorador de Internet por defecto del ordenador del usuario. **Figuras 6.6.17 y 6.6.18**



Figuras 6.6.17: Selección de ayudas.

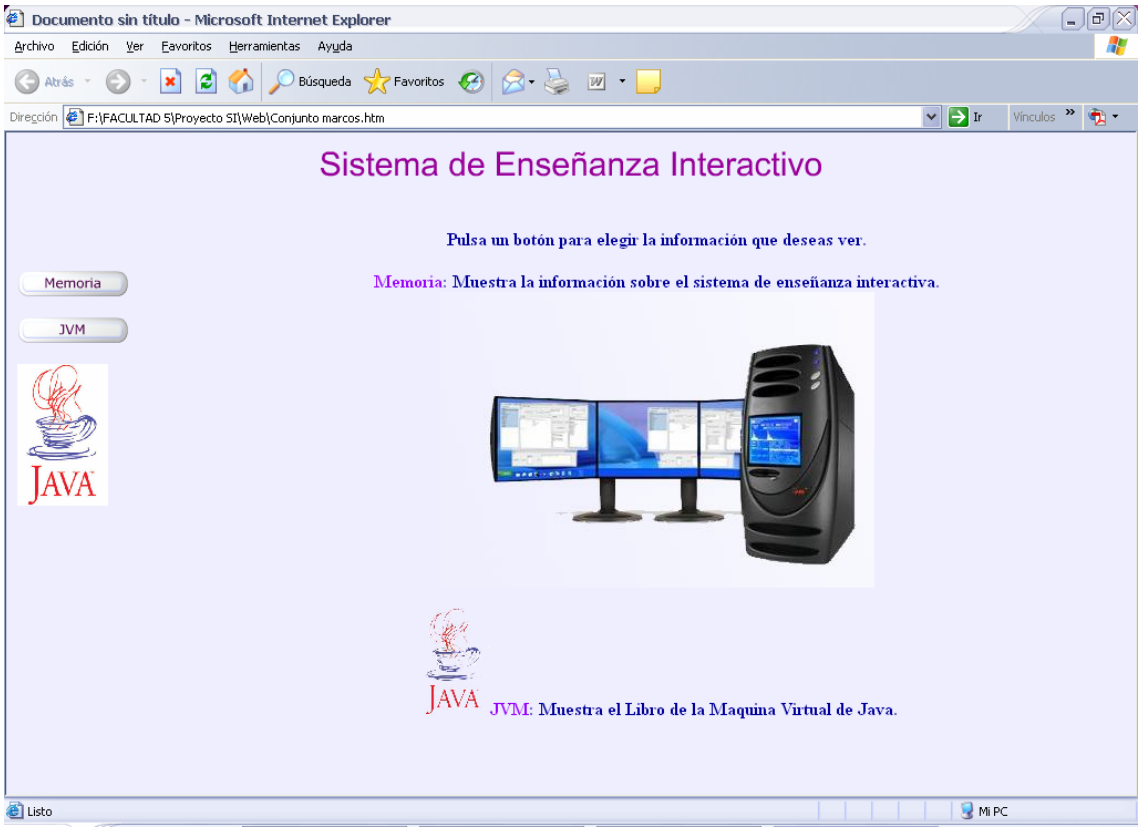
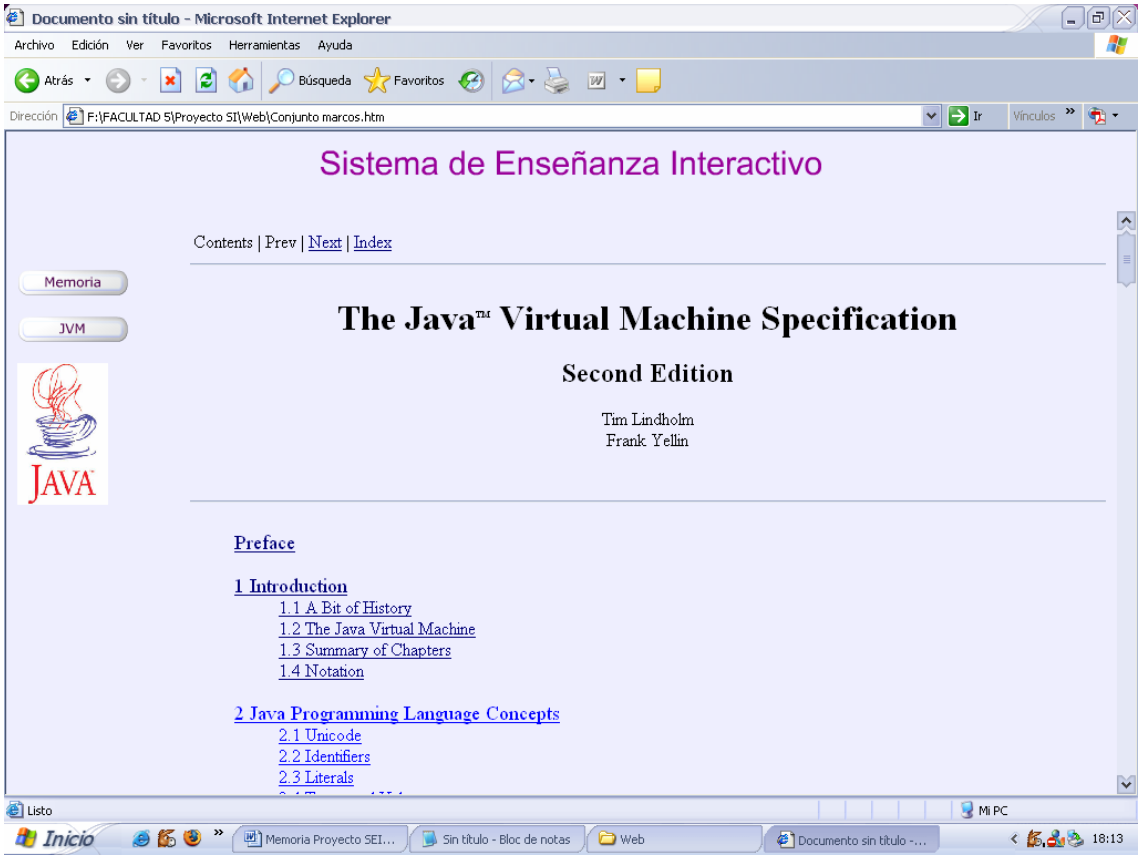


Figura 6.6.18: Ayuda en HTML



6.7 INSTALACIÓN

En este apartado se explicará qué hay que hacer para poder instalar el sistema, y comenzar a usarlo.

La instalación de la aplicación es sencilla simplemente hay que descomprimir el archivo"SEI_JVM.RAR" (Sistema de Enseñanza Interactivo de la JVM), incluido en el CD, en la carpeta deseada. Con ello tenemos la aplicación lista para su uso.

Para comenzar a usar cualquiera de los dos módulos:

- Editor de ejercicios y casos.
- Módulo de aprendizaje.

Simplemente hay que hacer "doble click" con el ratón en los archivos ejecutables (.BAT) "Editor Ejercicios y Casos.bat" y "Módulo De Aprendizaje.bat" con ello se pondrá en marcha el edito de ejercicios y casos o el módulo de aprendizaje.

La primera vez que se ejecute cualquiera de las dos aplicaciones se preguntará la ruta del Java Jdk. En posteriores usos de la aplicación esta información no será solicitada a no ser que se haya cambiado de lugar dicho Jdk o hubiera algun problema con el mismo. El Java Jdk se incluye con el CD del proyecto.

CAPÍTULO 7

FASE DE MANTENIMIENTO

7. FASE DE MANTENIMIENTO

7.1 INTRODUCCIÓN

Esta fase es una de las más importantes del ciclo de vida de un producto SW. El desarrollo de una aplicación no termina cuando el producto está funcionando, si así fuera en seis meses o un año el sistema quedaría obsoleto.

Normalmente esta fase no la lleva a cabo el grupo de desarrollo, sino que suelen ser personas que no conocen el código. Es por esto por lo que una buena documentación permite facilitar el desarrollo de esta fase. El objetivo que se trata de abordar en este capítulo es pensar en posibles ampliaciones y explicar cómo se llevarían a cabo.

7.2 AMPLIACIONES DEL SISTEMA

7.2.1 Cambio de los interfaces gráficos

7.2.1.1 Introducción

Una de las motivaciones iniciales para la realización del proyecto era el desarrollo de nuestro sistema de aprendizaje interactivo como un módulo o una extensión del entorno JAVY (JavA taught Virtually) **[JAVY],** un aplicación basada en la metodología “Learning by doing” donde los estudiantes pueden aprender el lenguaje de la maquina virtual de Java **(Capítulo 4.3) [JVM]** utilizando un entorno metaforico 3-D que simula la JVM. **[JVM],** en esta concepcion inicial nuestro sistema era el encargado de proporcionar las ayudas que necesitase el alumno, y por lo tanto la representacion grafica no debiamos tratarla nosotros. Finalmente decidimos hacer un sistema mas independiente que no se basase en la representacion grafica de JAVY sino que tuviese su propia interfaz pero que posteriormente puediese traladarse sin mucha modificacion para usarse con JAVY o con

cualquier otra solucion grafica distinta de la que proponemos inicialmente nosotros. En los siguientes puntos se explica el patron modelo vista controlador, que hemos usado para facilitar estos cambios de interfaz, y que pasos explicitos habria que seguir para realizarlos.

7.2.1.2 Modelo vista controlador (MVC)

El principal objetivo de la arquitectura MVC es aislar tanto los datos de la aplicación como el estado (modelo) de la misma, del mecanismo utilizado para representar (vista) dicho estado, así como para modularizar esta vista y modelar la transición entre estados del modelo (controlador). Las aplicaciones MVC se dividen en tres grandes áreas funcionales:

- Vista: la presentación de los datos
- Controlador: el que atenderá las peticiones y componentes para toma de decisiones de la aplicación
- Modelo: la lógica del negocio o servicio y los datos asociados con la aplicación

El propósito del MVC es aislar los cambios. Es una arquitectura preparada para los cambios, que desacopla datos y lógica de negocio de la lógica de presentación, permitiendo la actualización y desarrollo independiente de cada uno de los citados componentes.

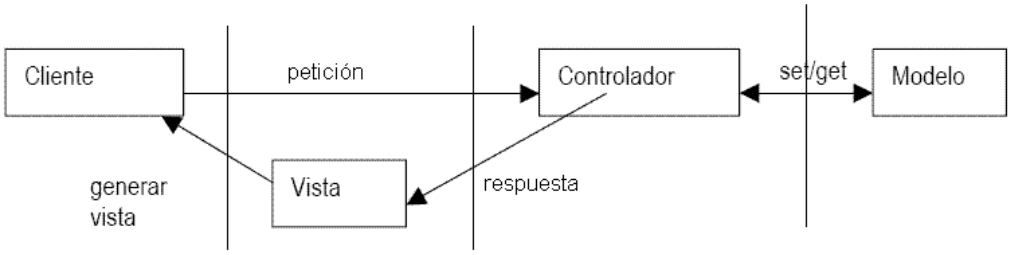
El MVC consta de:

- Una o más vistas de datos, en nuestro caso las representaciones con cajas y áreas de texto, listas desplegables, botones y demás componentes de Java Swing[3].
- Un modelo, el cual representa los datos y su comportamiento, en nuestro caso los ejercicios, los casos y las ayudas que se están resolviendo, creando o modificando.

[3] http://java.sun.com/docs/books/tutorial/

- Un controlador que controla la transición entre el procesamiento de los datos y su visualización, en nuestro caso las clases encargadas de reaccionar a los eventos que generan las acciones de los usuarios.

A continuación mostramos un esquema de este m



7.2.1.3 Cambio de interfaz del módulo de aprendizaje

Este módulo es el que estaba pensado para estar dentro de JAVY y por lo tanto hemos buscado que fuese fácilmente modificable, estos serian los pasos a seguir:

- Reescribir por completo la clase VentanaAlumnoGUI que contiene las definiciones de las componentes graficas.
- En la clase VentanaAlumno, que hereda de VentanaAlumnoGUI, y contiene las funcionalidades asociadas a las componentes graficas (hace el papel de controlador del MVC), ajustar el código a los cambios que se hayan realizado en la súper-clase.
- Extender la clase abstracta SelectorEjercicios, que, como el nombre indica, hace las funciones de cargar los ejercicios.

7.2.1.4 Cambio de interfaz del módulo de aprendizaje

Este módulo es más complejo y por lo tanto seria algo más incomodo realizar este cambio, estos serian los pasos a seguir:

- Al igual que con el módulo de aprendizaje cambiar VentanaProfesorGUI con la nueva representación grafica y ajustar los cambios en VentanaProfesor.
- Cambiar AyudaCasosGUI, que es la clase encargada de representar las ayudas que se incluyen en cada caso.
- Cambiar CodigoCompiladoGUI, que es la clase que muestra el código compilado (la solución) de un ejercicio o un caso.
- Cambiar CodigoErroneoGUI, que es la clase encargada de mostrar el código erróneo en el editor de casos.
- Cambiar SelectorCaso y SelectorAyuda, que son las clases se encargan de seleccionar a la hora de abrir casos y ayudas respectivamente.

7.2.2 Nuevo domino de conocimiento.

7.2.2.1 Introducción

Hay que distinguir dos procesos para realizar esta ampliación, por un lado es necesario modificar la herramienta para convertirla en framework y por otro añadir un nuevo conocimiento.

7.2.2.2 Framework

Un framework es una aplicación reusable, semicompleta que puede ser especializada para producir nuevas aplicaciones.

Añadir un nuevo conocimiento implica cambios en todo el sistema, esto implica prácticamente la creación de una nueva aplicación. Por ello nos planteamos la posibilidad futura de convertir la aplicación en un framework.

Esto daría todas las facilidades para añadir un nuevo dominio de conocimiento:

- Opción para **añadir una nueva ontología**. Por consiguiente toda la gestión que hay detrás de esta opción, como es la capacidad de almacenamiento, acceso, modificación y eliminación de dicha ontología.

- Opción para **crear una nueva base de casos**, que razone sobre la nueva ontología. Incluyendo todo su proceso de gestión, así como la creación de nuevas funciones de similitud.

- Opción para **modificar el interfaz gráfico del editor de ejercicios**. Los ejercicios de cada tema a enseñar poseen estructuras diferentes, por tanto el editor para esos ejercicios también varía.

- Opción para **modificar el interfaz gráfico del módulo de aprendizaje del alumno**. Al igual que ocurre con el editor de ejercicios, también los pasos para resolver un problema dependen directamente del ejercicio.

El acceso para un usuario, a cualquiera de las funciones detalladas anteriormente, debe hacerse mediante un interfaz gráfico sencillo y manejable. Por ello otra tarea importante es la creación del interfaz que de soporte a las funciones necesarias para incorporar un nuevo dominio de conocimiento.

Puesto que un alumno puede aprender sobre distintas materias, los perfiles de usuario también deben cambiar, incorporando información de

nivel y temas realizados para cada dominio de conocimiento. Por tanto el sistema debe poseer los mecanismos necesarios para la gestión de la nueva información.

Puesto que los ejercicios son concretos del dominio de conocimiento al que pertenecen, la jerarquía de ejercicios no debe de ser única sino que habrá una por cada tipo de conocimiento. Por consiguiente la aplicación debe poseer los métodos necesarios para llevar a cabo la gestión de las jerarquías de ejercicios.

Una vez terminado con todo el proceso, se puede pasar a incorporar un dominio de conocimiento, para ello se puede usar el de la JVM, y comprobando que el framework funciona correctamente.

7.2.2.3 Añadir nuevo conocimiento al framework.

Realizar este proceso requiere que se haya hecho previamente el proceso detallado en el apartado anterior **(7.2.2.2)**.

Los pasos a seguir son los siguientes:

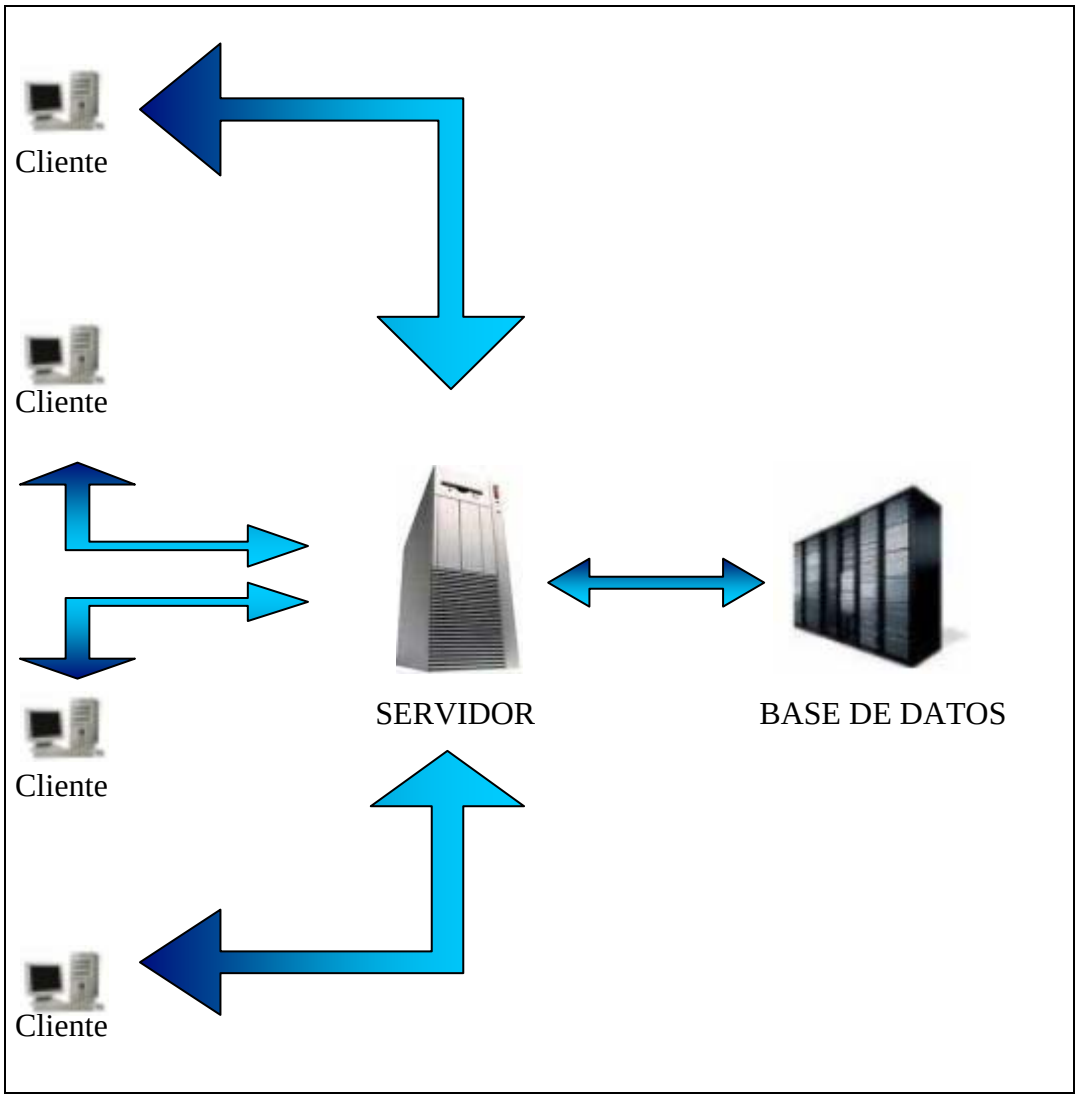
- Creación de la nueva ontología. Si se desea automatizar este proceso primero hay que extraer el conocimiento y luego transformarlo a formato OWL mediante el conector java-OWL explicado en el apartado 3.2.4.

- Diseño del interfaz gráfico que de soporte al nuevo tipo de conocimiento. Esta tarea se realiza mediante las funcionalidades que aporte el framework para el interfaz de aprendizaje del alumno y del editor de ejercicios.

- Crear la base de casos que permita generar ayudas que faciliten y expliquen la resolución de un determinado ejercicio.

7.2.3 Sistema distribuido.

Una ampliación muy deseable es que el sistema pueda ser ejecutado en red. En este apartado mostramos los pasos necesarios para hacer que la aplicación se ejecute en red con una arquitectura en 3 capas, cliente-servidor-BBDD



El diseño modular facilitará las tareas para desarrollar esta ampliación.

7.2.3.1 Cliente

Se entiende por cliente aquellas aplicaciones que acceden remotamente al sistema. Es conveniente ejecutar el máximo código en el cliente para liberar de trabajo al servidor, de esta manera obtenemos un mejor rendimiento.

El módulo de aprendizaje del alumno es un candidato perfecto para integrarse en el cliente. Esto aporta una ventaja muy importante a la aplicación, el sistema aprenderá de los errores cometidos por cualquier alumno, y éstos servirán para generar una ayuda más sofisticada ante los errores de otros alumnos.

El editor de ejercicios también puede integrarse como cliente. El problema de que una persona tenga que editar todos los ejercicios queda oculto. Los ejercicios que edite un profesor quedan almacenados en el sistema, pudiendo ser usados por cualquier usuario. Pero surge un problema, la *gestión de acceso y modificación* a los ejercicios, hay que añadir *seguridad* para que los ejercicios almacenados sean correctos.

El módulo gestor de ayudas no puede ser incorporado en su totalidad en el cliente, porque trabaja con información común a los clientes.

Un factor muy importante es la comunicación entre las diferentes capas de la arquitectura de red, este aspecto se detalla en el apartado 7.2.3.3 que hace referencia al servidor.

7.2.3.2 Base de datos

La base de datos almacenará toda la información que precise el sistema y que no sea de carácter temporal.

- Ontologías de los distintos dominios de conocimiento capaz de enseñar el sistema.

- Jerarquías de ejercicios, una por cada dominio de conocimiento.

- Perfiles de usuario.

- Base de casos del sistema CBR del módulo gestor de ayudas.

7.2.3.3 Servidor

Es el encargado de controlar la gestión de todos los clientes del sistema, así como los accesos a la base de datos. La comunicación entre las tres capas (cliente-servidor-BD) se realiza entorno al servidor. En una aplicación distribuida es muy importante definir un protocolo de comunicación que defina los elementos que se transfieren y el orden.

7.2.3.3.1 Comunicación cliente-servidor.

El cliente puede ser o bien un profesor o bien un alumno, de esto depende mucho la comunicación entre cliente y servidor.

- **Alumno**. Lo primero es identificarse. Por tanto el primer punto de comunicación es para obtener el perfil de usuario del alumno. El siguiente paso es proponerle un ejercicio acorde con su perfil, lo que conlleva otro punto de comunicación. Para la solicitud de ayuda hay dos opciones, una es tener la base de casos del dominio concreto en el cliente, que no implica comunicación, y la otra es que esté en la base de datos, que se realizará una conexión por cada ayuda solicitada. Por último hay que actualizar el perfil de usuario si resuelve el ejercicio.

- **Profesor**. En este caso la comunicación es muy sencilla, primero tiene que identificarse como profesor para que le estén permitidas ciertas acciones. Editar un ejercicio, o crear un nuevo caso para el sistema gestor de ayudas, no implica comunicación a excepción

del momento en que desee guardarlo. Modificar un ejercicio (si se permite) implica comunicación al acceso y en el almacenamiento del mismo.

CAPÍTULO 8
CONCLUSIÓN

8. CONCLUSIONES

La aplicación final es un sistema de enseñanza interactivo, basado en técnicas CBR **[CBR]** para la gestión de ayudas. Dos son los aspectos claves en el sistema: la gestión de ayudas y el dominio de conocimiento.

La gestión de ayudas es la parte inteligente del proyecto, alberga los dos sistemas CBR (CBR textual y CBR tradicional). Cuando el alumno lo solicita se le proporciona la ayuda que en circunstancias similares permitió resolver un problema parecido a otro alumno, el término "circunstancias similares" hace referencia a la técnica CBR, que almacena circunstancias (o casos), y busca mediante una función de similitud aquellas que más se parecen a la actual. Además el alumno puede hacer una pregunta concreta directamente al sistema el cual responde utilizando la documentación textual del dominio del conocimiento que se esta tratando de enseñar. Todo esto permite orientar a un alumno durante la resolución de un ejercicio, y por tanto, permitir que un alumno no se quede bloqueado en la resolución del mismo.

El dominio de conocimiento que enseña el sistema es acerca de la maquina virtual de JAVA **[JVM]**, por tanto, permite aprender el repertorio de instrucciones de la JVM mediante la resolución de ejercicios ("Learning by doing" **[LBD]**) y el funcionamiento de cada instrucción. El conocimiento se ha extraído de un libro en HTML, y se ha desarrollado de forma automática una ontología que lo define.

La identificación a los usuarios se consigo gracias a los perfiles de usuario, que además sirven de base al sistema para la proposición de un ejercicio acorde con el nivel de conocimiento del alumno.

Esta herramienta posee un gran potencial de desarrollo, una vez se realicen las ampliaciones que se detallan en el **capítulo 7.2** el sistema será capaz, no sólo de aprender con gran rapidez, sino de ser una herramienta eficaz para el desarrollo de cursos a distancia. La capacidad de aprender distintos dominios de conocimiento aporta una gran versatilidad a la herramienta, y el hacer la aplicación distribuida permite no

sólo ampliar los dominios de conocimiento a gran velocidad, sino también mejorar las ayudas propuestas a los alumnos.

Por tanto, se han conseguido la mayoría de los objetivos iniciales propuestos y además con un resultado muy satisfactorio. Este proyecto nos ha servido para aprender a fondo muchas tecnologías que desconocíamos y nos ha enseñado como trabajar en equipo tanto a nivel personal como a nivel de organización, diseño, e implementación.

BIBLIOGRAFIA

BIBLIOGRAFIA

[JVM] The Java™ Virtual Machine Specification
http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html

[SIP] Departamento de Sistemas Informáticos y Programación
http://www.ucm.es/info/dsip/

[C++] Tutorial de C++:
http://www.cplusplus.com/doc/tutorial

[JColibri] JColibri:
http://sourceforge.net/projects/jcolibri-cbr

[GAIA] Group of Artificial Intelligence Applications:
http://gaia.fdi.ucm.es

[JAVY] Java taught virtually:
http://gaia.fdi.ucm.es/grupo/projects/javy/index.html

[ONT] Ontologia:
http://es.wikipedia.org/wiki/Ontolog%C3%ADa_%28Inform%C3%A1tica%29

[UML] Unified Modeling Language
ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/intro_rdn.pdf

[JAVA] Java
http://java.sun.com/

[Protégé] Protégé
http://protege.stanford.edu/index.html

[OWL] Modificación de un fichero OWL(Creación automática de ontología)
http://protege.stanford.edu/plugins/owl/api/guide.html

[JENA] Jena
http://jena.sourceforge.net/

http://protege-owl.sourceforge.net/javadoc/edu/stanford/smi/protegex/owl/jena/Jena.html

[RR] Rational Rose
http://searchvb.techtarget.com/sDefinition/0,,sid8_gci516025,00.html

[LBD] Learning By Doing
http://www2.glos.ac.uk/gdn/gibbs/index.htm

[Riesbeck and Schank, 1989] *Inside Case-based Reasoning:* New Jersey, Lawrence Erlbaum, 1989.Riesbeck, C. K. and Schank, R.C.

[Kolodner & Leake 1997] *A Tutorial Introduction to Case-Based Reasoning. To appear in Case-Based Reasoning: Experiences, Lessons, and Future Directions.* J. Kolodner and D. Leake, , D. Leake (ed.), AAAI Press, 1996

[Nelson & Richard Hollingham] *How to Clone the Perfect Blonde: Making Fantasies Come True with Cutting-Edge Science by*Sue Nelson, Richard Hollingham, Ebury Press

GLOSARIO DE TÉRMINOS

GLOSARIO DE TÉRMINOS

A

Alumno: Usuario que trata de aprender un dominio de conocimiento

C

ComboBox: Lista desplegable utilizada en los interfaces gráficos en JAVA.

Case Base Reasoning (CBR): En Español razonamiento basado en casos, es un modo natural de razonamiento de los seres humanos, es decir, los seres humanos nos basamos en experiencias pasadas para encontrar solución a los problemas que se nos plantean

I

Ingeniería del software: rama de la ingeniería que crea y mantiene las aplicaciones de software aplicando tecnologías y prácticas de las ciencias computacionales, manejo de proyectos, ingeniería, el ámbito de la aplicación, y otros campos.

Inteligencia Artificial: La inteligencia artificial (abreviado IA, en inglés AI) es definida como la inteligencia exhibida por una entidad artificial. Generalmente se asume que dicha entidad o sistema sea un computador. (http://es.wikipedia.org/wiki/Inteligencia_artificial)

J

Java: Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución, y un conjunto de librerías estándar que ofrecen funcionalidad común. (http://es.wikipedia.org/wiki/Plataforma_Java).

JVM: Java Virtual Machina, máquina virtual de Java. *Véase Maquina virtual*

Jena*:* es un framework de Java para la construccion de aplicaciones de webs semánticas.

L

Learning by doing: paradigma basado en el aprendizaje a partir de la realización de ejercicios.

M

Maquina virtual: En informática una máquina virtual es un software que crea un entorno virtual entre la plataforma de la computadora y el usuario final, permitiendo que este ejecute un software determinado. (http://es.wikipedia.org/wiki/M%C3%A1quina_virtual)

P

Profesor: Usuario con conocimiento del domino que trata de enseñar.

Protégé: es un editor de ontologías y un framework de bases del conocimiento gratuito y de codigo abierto.

R

Rational Rose: Rational Rose es una herramienta de diseño de UML orientado a objetos pensada para el modelado visual y la construcción de componentes de aplicaciones sofware de nivel empresarial.

S

Software (SW): es el conjunto de instrucciones que permite al hardware de la computadora desempeñar trabajo útil. En las últimas décadas del siglo XX, las reducciones de costo en hardware llevaron a que el software fuera un componente ubicuo de los dispositivos usados por las sociedades industrializadas.
(http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software)